Universidad Carlos III de Madrid

Escuela politécnica superior

Departamento de Informática

Ingeniería en Informática

Proyecto Fin de Carrera

Editor de Gramáticas Relajadas para el Procesamiento de Lenguaje Natural

Autor: Enrique Sánchez Sotés

Tutores: Francisco Javier Calle Gómez

María Dolores Cuadra Fernández

Agradecimientos:

En primer lugar me gustaría agradecer todo en esta vida a mis padres, personas que han sabido educarme y hacer de mi un buen hombre con defectos y virtudes. Nunca me ha faltado apoyo por vuestra parte, siempre que he necesitado algo me lo habéis proporcionado. Aún resuenan en mi cabeza frases como "Algún día nos lo agradecerás…" o "lo hacemos por tu bien.". Hoy es el día en el que se cumplen. Hoy os agradezco todo lo que habéis hecho por mí y cómo lo habéis hecho. Os lo debo todo.

Doy gracias a una persona que, sin ella, no hubiese reunido nunca el valor suficiente como para seguir el día a día. El aliciente que ha supuesto, que día sí y día también, durante unos fantásticos siete años, madrugue, atienda y estudie. Tú más que nadie ha supuesto un pilar principal en toda esta etapa. Me has dado lo que más necesitaba: cariño, comprensión, amistad y amor. Por esos días en los que me explicabas trigonometría (aún sigo portando en la cartera las fórmulas), aquellos que hacíamos que estudiábamos y por aquellos en los que los agobios de las prácticas no nos dejaron hacer cualquier otra cosa. Gracias Espe. Todos los días han sido, son y serán maravillosos a tu lado.

Por otro lado es menester agradecer al fantástico grupo de amigos que son los Cebollitas (en todas sus variantes) todos esos buenos ratos que hemos pasado. Gracias por todos esos partidos de volley en los que descubrí que tenía una lesión en el hombro. Gracias por todas esas jornadas ociosas que han tenido lugar, todas esas reuniones en las que las risas eran predominantes ante cualquier otra circunstancia. Gracias C, H, L, K, K, A, A, D, J, S, M, E (si, otra vez tú), B (o M) y un largo etcétera.

Debo agradecer también a mis tutores Javi y Loli todo el esfuerzo que han realizado durante este año. Gracias por confiar en nosotros, por darnos la oportunidad de formar parte del grupo LaBDA y de conseguir que este proyecto, y el trabajo diario que estamos realizado no sea una obligación sino un placer.

Por último y no menos importante, agradecer a la fantástica gente que forma parte del laboratorio, que nos acogieron desde el primer día que entramos por la puerta y nos hicieron sentir unos integrantes más. Gracias Dani, Adri, Leo, Pablo, Rebe, Juli y David.

Índice de contenidos

Índice de ilustraciones

1 Introducción

La expansión de las nuevas tecnologías incrementa la necesidad de mejorar las interfaces de comunicación entre humanos y ordenadores. Una de las mejoras más perseguidas es conseguir esta interacción sin que las personas tengan que adquirir previamente conocimientos específicos o realizar un aprendizaje técnico antes de manejar cualquier ordenador o programa informático. La interacción natural está enfocada a imitar el comportamiento de la interacción entre humanos.

Marco de trabajo y motivación

Con el fin de imitar el comportamiento humano es necesario estudiarlo previamente y extraer el conocimiento a partir de la comunicación entre dos personas. Este conocimiento extraído debe formalizarse de algún modo para que el sistema de interacción sea capaz de utilizarlo.

Es necesario distribuir todo el conocimiento que se obtiene de las interacciones en distintos modelos de conocimiento, cada uno de ellos especializado en un aspecto concreto de la interacción. Suelen clasificarse en función de la naturaleza de su conocimiento o el dominio al que hacen referencia y forman parte de la arquitectura cognitiva de un sistema de interacción natural.

En la Ilustración 1.1, se muestra un esquema de la arquitectura cognitiva de un sistema de interacción natural propuesta por (Calle F. J., 2004). En este esquema se pueden apreciar diferentes módulos, cada uno de ellos independiente entre sí pero relacionado con los demás.

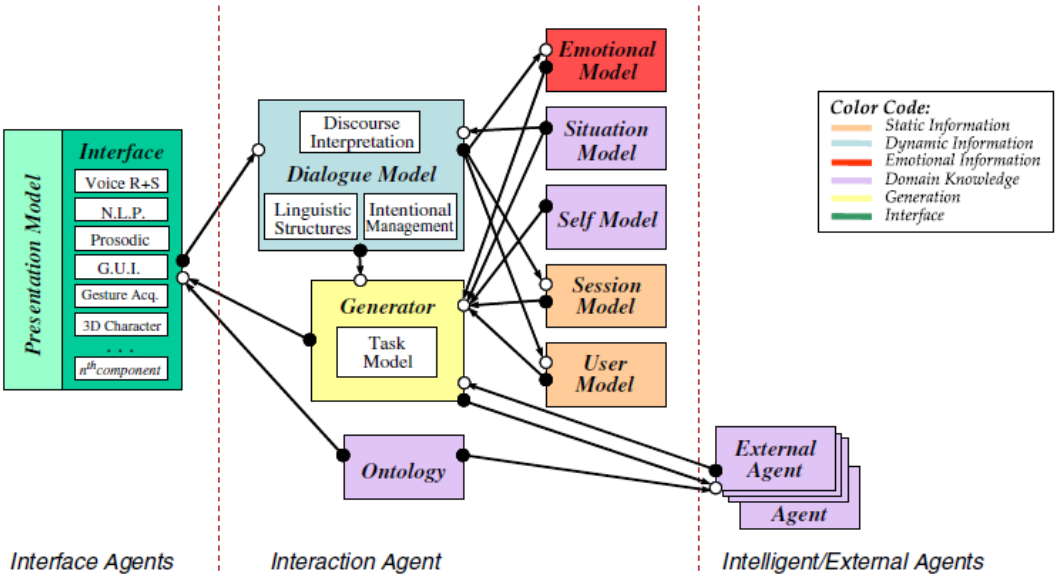


Ilustración 1.1: Arquitectura cognitiva

El Modelo de Presentación trabaja junto con los componentes de interfaz (reconocedor de voz, interfaz de usuario, avatar, etc.) con el propósito de coordinar todas las entradas de información en la fase de interpretación y todas las salidas en la fase de generación para así poder producir intervenciones completas y coherentes. Además ayuda a gestionar los turnos y manejar el estado de la interacción.

El componente Interfaz representa la cara visible para el usuario. Su función es recoger las entradas por parte del usuario y devolver las salidas por parte del sistema. Este módulo puede ser implementado mediante diversos tipos de interfaces, como puede ser un reconocedor de gestos, un avatar 3D, un reconocedor de voz, un procesador de lenguaje natural, etc. El propósito final de este proyecto es generar una herramienta que permita la adquisición de conocimiento con el fin de alimentar un módulo de este componente, concretamente el procesador de lenguaje natural.

El Modelo de Diálogo fija el estado de la evolución de la interacción y define el comportamiento interactivo del sistema. Para ello, procesa las acciones comunicativas del interlocutor y desarrolla las suyas propias.

El Generador de Discurso, que contiene un Modelo de Tareas, es el encargado de gestionar las funciones o tareas que el sistema debe llevar a cabo en cada momento de la interacción (por ejemplo, consultar un reloj interno o conectarse a internet), y producir el contenido de las intervenciones del sistema.

La Ontología se encarga de gestionar el conjunto de conceptos y sus relaciones que serán susceptibles a ser referenciados a través de términos dentro de un dominio de interacción.

El resto de modelos se encargan de gestionar otros aspectos de la interacción, estos son: Modelo Emocional (opera sobre información referente a emociones), Modelo de Situación (define las circunstancias en las que se enmarca la interacción), Automodelo (gestiona metas individuales del sistema forjando una personalidad), Modelo de Sesión (almacena el punto de partida y la evolución de la interacción) y Modelo de Usuario (maneja los datos del interlocutor).

Todos estos componentes forman parte de una arquitectura multi-agente, lo que implica que se ejecutan en procesos diferentes, actúan de modo independiente y colaboran entre ellos persiguiendo un mismo objetivo.

Para que esta arquitectura consiga realizar una interacción lo más natural posible se requiere conocimiento específico de un dominio concreto. Este conocimiento proviene de un corpus de un determinado dominio que es adquirido a través de la observación de interacciones entre humanos. En esta observación no sólo se tendrán en cuenta los diálogos que suceden, sino que además se recogerán datos gestuales, cambios de tono, expresiones faciales, etc.

Toda la información del corpus adquirido debe ser analizada y anotada por expertos en lingüística, concretamente pertenecientes al área de la pragmática. De esta forma, se hace posible que la información contenida en el corpus pueda ser interpretada por el sistema y además sea utilizada para conseguir imitar el comportamiento humano.

Objetivos

Un procesador de lenguaje natural tiene como tarea extraer expresiones por parte del interlocutor, procesar estas expresiones y tratarlas y formalizarlas de tal manera que un sistema (informático en este caso) sea capaz de comprenderlas y operar con ellas. Para ello, contiene un motor que infiere el conocimiento del lenguaje natural presente en su base de conocimiento. El procesador del lenguaje natural propuesto en la arquitectura cognitiva de la Ilustración 1.1 realiza las siguientes tareas:

- Interpretación: a partir de una expresión cualquiera, el motor de inferencia genera uno o varios actos comunicativos según corresponda.

- Generación: a partir de uno o varios actos comunicativos proporcionados por el sistema, el motor de inferencia generará una o varias expresiones en lenguaje natural.

Este proyecto ha sido concebido con el fin de facilitar la tarea de extracción de información y el almacenamiento en una base de conocimiento específica. Esta información procede de Corpus de diálogo. Para ello se plantea una interfaz gráfica en la cual el usuario podrá editar las gramáticas de todas las expresiones que formen parte de un corpus. A su vez se plantea el diseño e implementación de una base de datos en la cual se almacenará todos los datos extraídos por la herramienta a lo largo de su utilización.

La tarea de extracción de gramáticas la lleva a cabo un experto lingüista especializado en pragmática, pero no necesariamente familiarizado con la informática. Por ello, uno de los objetivos más importantes de esta herramienta es realizarla lo más amigable e intuitiva posible.

2 Estado del arte

En este punto se describirán los conceptos fundamentales para la comprensión del proyecto los cuales abarcan un gran abanico de disciplinas, incluyendo en gran parte conceptos lingüísticos y el cómo la informática aplica técnicas para la resolución de problemas de este tipo. Se incorporará en este apartado definiciones sobre los conceptos técnicos que maneja el editor, una panorámica sobre trabajos similares que se están desarrollando, los cuales provienen de distintos grupos de investigación dentro del área de la interacción natural, y específicamente en el procesamiento de lenguaje natural.

Fundamentos teóricos

Este proyecto contiene una amplia carga de conceptos teóricos asociados a la lingüística que alguien que no esté familiarizado con el ámbito no comprendería. Para ello en este apartado se pretende aclarar todos ellos. La mayoría de la información que se presentará a continuación parte de distintos estudios que lingüistas y filósofos han investigado a lo largo de estos años en torno a la comunicación humana.

Existen muchos conceptos que rodean el ámbito de este proyecto y otros que éste maneja. Es necesaria una comprensión de todos ellos. Dado el ámbito de la interacción natural no se puede negar que un elemento indispensable de ésta es el lenguaje verbal, que es un lenguaje hablado o escrito por los humanos para satisfacer necesidades y propósitos de comunicación.

2.1.1 Actos de habla y actos comunicativos

La teoría de los actos de habla, tal y como la formuló el filósofo del lenguaje J. L. Austin es una de las consecuencias de la filosofía del lenguaje peculiar. Aparece formulada en su obra póstuma *How to Do Things with Words* (Austin, 1962). Dicha teoría es el arranque de uno de los enfoques más populares en el área de la pragmática.

Un acto de habla o acto ilocutivo es un tipo de acción que involucra el uso del lenguaje natural y está sujeto a un cierto número de reglas convencionales. Austin presenta una propuesta de agrupación de estos actos:

- Acto locutivo: Es la acción de realizar una expresión oral, lo que se dice.
- Acto ilocutivo: Es la intención o finalidad concreta del acto de habla.
- Acto perlocutivo: es el (o los) efecto(s) que el enunciado produce en el receptor en una determinada circunstancia.

Se podría concluir de esto que en toda acción comunicativa oral estarán presentes estos tres tipos de actos de forma simultánea. También existe comúnmente otra división de actos de habla: actos directos y actos indirectos. Los actos directos son aquellos en los que el aspecto ilocutivo literal de la oración coincide con el aspecto ilocutivo real, es decir, se expresa exactamente la intención del hablante. Los actos indirectos son aquellos en los que esta coincidencia no se produce, es decir, son las frases en las que la finalidad es distinta de lo que se expresa directamente.

En toda interacción humana no solo intervienen acciones de tipo verbal (tanto oral como escrito). Por ello con el fin de conseguir una mayor generalidad de información y por tanto, recoger factores no verbales de la interacción se introduce el concepto de Actos Comunicativos como extensión de los anteriores. Los actos comunicativos permiten unificar cualquier expresión (verbal y no verbal) en la misma representación y con el mismo formato, por tanto permitirán representar gestos, pausas, solapamientos y demás conocimiento prosódico que todo humano identifica y representa en sus interacciones.

2.1.2 Dominio de interacción

Dominio de interacción es un concepto fundamental en cualquier sistema de interacción natural. Se define dominio como un área de experiencia y conocimiento en alguna actividad del mundo real. Interacción es el conjunto de acciones e intercambios de información que se suceden entre varios sujetos, objetos funciones, etc. En el caso particular de este proyecto la interacción a la que se refiere este documento constantemente es la comunicativa. Un dominio de interacción determina, por tanto, el

conocimiento necesario para que se produzcan los intercambios entre ambos interlocutores y que éstos sean satisfactorios para un conjunto de fines.

De esta forma queda aclarado el dominio sobre el cual se desarrolla una comunicación satisfactoria. Es necesario discernir todo dominio de interacción entre sí, ya que no todas las personas poseen experiencia y conocimiento para interactuar en determinadas áreas. Todo individuo está capacitado para conversar en multitud de dominios pero nunca en todos, siempre llega un punto en el cual se requiere más conocimiento para ampliar el conjunto de dominios ya que sin éste conocimiento, un individuo en un dominio extraño será incapaz de transmitir opiniones, actuar o interpretar.

Idénticamente a las personas, sucede lo mismo en los sistemas de interacción natural. Es necesario limitar el dominio para que el sistema se desenvuelva satisfactoriamente (y no siempre se consigue). Por ello, para alimentar a estos sistemas, este proyecto deberá de ser capaz de suministrar conocimiento a cada dominio de interacción concreto mediante una interfaz general.

Un dominio de interacción contendrá un conjunto de escenarios que representan distintos tipos de conversaciones que se puede dar entre el usuario y el sistema de interacción con un determinado objetivo o una tarea específica. Cada uno de estos escenarios estará compuesto por varios diálogos que demuestran fielmente el cómo un humano realizará con un experto del dominio dichas tareas.

Además, un diálogo proveniente de cualquier interacción está compuesto por intervenciones, que identifican los distintos turnos por parte de los participantes sin que presenten interrupciones por parte del segundo hablante. Cada una de estas intervenciones puede constar de una o varias frases así como de distintos símbolos no verbales.

2.1.3 Gramáticas relajadas

La pretensión de este proyecto no es lograr analizar semánticamente las frases sino obtener conocimiento pragmático individual a nivel de expresión. Por ello el análisis que se realiza es a nivel más general que los análisis sintácticos o semánticos. Se presentan por tanto, estructuras gramaticales arbóreas que constan de elementos identificados, enfocados a obtener la información pragmática de cualquier expresión en

cualquier lenguaje, ya que este análisis es totalmente independiente de la gramática o morfología propias del idioma.

Una gramática relajada con enfoque pragmático consta de los elementos que se definirán a continuación, posteriormente se incluye un ejemplo de uso de los más importantes de estos.

- Patrón raíz: Es el nodo principal del árbol, el elemento principal o el punto de inicio de la gramática. Es el identificador principal para una expresión analizada. Puede contener cualquier número de los elementos inferiores así como uno o varios actos comunicativos asociados a la expresión que se esté analizando.

- Sub-patrón: Elemento idéntico a un patrón, con la salvedad de que ocupa lugar en niveles inferiores a este. Puede contener por tanto otros sub-patrones y cualquier número de los siguientes elementos. Como el patrón, es necesario que este elemento contenga al menos un acto comunicativo.

- Token: Elemento similar a los patrones, pero éste no contendrá ni sub-patrones inferiores ni actos comunicativos asociados. Pero sí que es posible que un token contenga tokens inferiores. De esta forma se permite un análisis más cómodo permitiendo dividir expresiones extensas en fragmentos menores. Los tokens son los que almacenan los verdaderos terminales de la gramática, las palabras de la expresión. Posteriormente se realizará una descripción en detalle de este complejo elemento.

- Variable: Elemento que define información contextual de la expresión, esta información será general y perteneciente a un tipo (dominio) concreto de variable. Actualmente estos tipos son: cantidad, hora y fecha. En un futuro, cuando el sistema COGNOS esté integrado con la ontología, se ampliarán estos dominios permitiendo de esta forma representar cualquier concepto en cualquier dominio de interacción.

- Comodín: Elemento que representa cualquier cosa, cuando una expresión contiene comodines se quiere decir que las partes marcadas como tal no son relevantes en la conversación y por tanto, omisibles. Se permite incluir

restricciones en cuanto al número de palabras de las que puede constar el comodín.

Como se ha comentado, un patrón raíz y sus elementos inferiores representan un análisis de una expresión los cuales pueden ser otros sub-patrones, tokens, variables y/o comodines, los sub-patrones pueden contener a su vez otros sub-patrones y los tokens, otros tokens. Se forma una estructura arbórea similar a la de las gramáticas libres de contexto.

Un posible ejemplo es el que se aprecia en la Ilustración 2.1 en el cual se muestra el árbol de elementos que representa el análisis de la frase: "Buenos días, quiero una cerveza". Esta frase contiene dos actos de habla distintos, por tanto para realizar un análisis correcto y enfocado a la reutilización, se podrá subdividir el saludo inicial del resto de la frase, editándolo como sub-patrón. Este contiene un único token (Token 1) que está compuesto por dos palabras (buenos y días). Los dos tokens restantes presentes en el análisis únicamente están compuestos por una palabra (quiero y cerveza respectivamente). La expresión contiene además una variable que indica la cantidad de cervezas que quiere el hablante. Se trata sin duda de un elemento variable de dominio de cantidad y será analizado como tal.

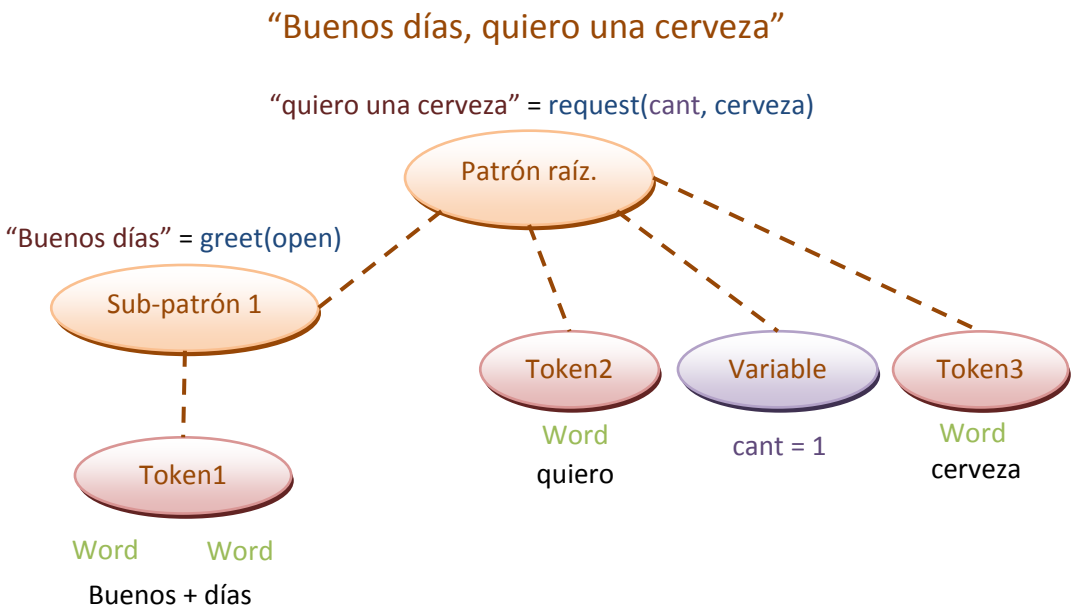


Ilustración 2.1: Ejemplo de árbol de derivación

En este punto es necesario introducir el concepto de descripción como ampliación a la definición de token que se presentó anteriormente. Un token dispone de una colección de descripciones. Son llamadas descripciones de un token a aquellas que reflejan las distintas posibilidades de formación que un token dispone. Un ejemplo sencillo de esto es, en la Ilustración 2.1, el token llamado "token2" contendrá una descripción con la palabra quiero. En un primer análisis ésta será su única descripción, pero a medida que se vayan analizando expresiones del mismo dominio, probablemente aparezcan frases como "deseo un refresco" ó "me gustaría una copa". Las expresiones "quiero", "quisiera", "deseo", "me gustaría", etc. formarían en este análisis las descripciones concretas del token "token2".

2.1.4 Trabajo previo

Uno de los múltiples objetivos de obtener patrones de expresiones de lenguaje natural es para permitir a un sistema interpretar, conocer y comprender las intenciones de las expresiones. Para interpretar una expresión de lenguaje natural se necesita, además de un sistema que conozca las estructuras de almacenamiento y algoritmos de encaje, una gran cantidad de conocimiento almacenado en forma de patrones.

Para el desarrollo del editor se dispone de un procesador de lenguaje natural previamente desarrollado y proporcionado por uno de los miembros del grupo (LaBDA). Este procesador está implementado en java y cuenta con estructuras de información propias para los patrones de expresiones las que son usadas tanto para interpretar como para generar lenguaje natural. El procesador proporcionado está basado en uno previamente programado en Prolog y usado en varios proyectos como (IntegraTV-4all) ó ADVICE.

Este motor de inferencia de lenguaje natural parte de una colección de patrones y una expresión, realiza un recorrido por la estructura de todos ellos y obtiene el o los que encajen. Una vez localizado el patrón con encaje óptimo (o el primer encaje en caso de no disponer de mecanismos avanzados) se obtiene los actos comunicativos que se editaron para ese patrón. Para obtener el patrón que encaje existen muchos algoritmos, conocidos algoritmos de búsqueda y de recorrido de árboles, en este caso se recorren los nodos terminales en profundidad verificando su encaje.

Un patrón encaja con una expresión dada si todos sus elementos se encuentran en la posición correcta y se adecuan a las partes de la expresión. Cada elemento dispone de un mecanismo de encaje diferente:

- Token: Una expresión encaja con un elemento token si alguna de las descripciones del token encaja a su vez con la expresión. Las palabras encajan si son literalmente idénticas.
- Variable: Una expresión encaja en una variable si esta coincide con la expresión regular que está asociada al dominio. Por ejemplo, la expresión "12:32" encaja con una variable de tipo hora.
- Comodín: Si el comodín tiene restricciones respecto a cantidad de palabras, una expresión encajará si se cumple dicha restricción. Si no tiene ninguna restricción, cualquier expresión encajará.
- Sub-patrón y patrón: Una expresión encaja con un patrón o un sub-patrón si todos sus elementos encajan.

Herramientas relacionadas

Existen en el marco del procesamiento de lenguaje natural multitud de tipos de análisis. Un análisis semántico está enfocado a obtener el significado de las palabras y expresiones, omitiendo el propósito o función de estas. Un análisis pragmático se centra principalmente en el uso de un lenguaje de interacción social y en los efectos que este uso pueda producir. Este proyecto pretende anotar expresiones con un enfoque más pragmático que semántico. La obtención del conocimiento pragmático de la comunicación es indispensable para que un modelo de diálogo sea capaz de conseguir imitar la comunicación humana.

Existen muchas herramientas relacionadas con el procesamiento de lenguaje natural, desde analizadores, traductores, ontologías, etc. Las herramientas más acordes al ámbito de éste proyecto y más populares son los editores de lenguaje natural basados en la edición de gramáticas libres de contexto (CFG). Por otro lado, existe una inmensidad de herramientas de etiquetado y análisis sintáctico, morfológico y semántico, pero es muy difícil encontrar interfaces gráficas para la anotación pragmática de expresiones individuales. La diferencia entre anotaciones semánticas y pragmáticas

puede no ser, en ocasiones, muy amplia, incluso sería posible una automatización partiendo de la primera. Por tanto, en este apartado se tomarán como referencia los mecanismos de anotación y almacenamiento de algunas herramientas de anotación.

La principal diferencia con estos sistemas es que, aunque suelen poseer un gran ámbito de trabajo (la mayoría no solo facilitan el anotado semántico y morfológico sino que también analizan o procesan información), no lo hacen a nivel de análisis que en ocasiones es necesario para permitir simular habilidades humanas.

2.2.1 General Arquitecture for Text Engineering: GATE

GATE (The University of Sheffield) es un conjunto de herramientas desarrollado en java realizado por la Universidad de Sheffield en 1996. En 2002 fue totalmente re-diseñado y se volvió a publicar. Es un sistema muy expandido, es usado en todo el mundo por la comunidad de investigación, compañías, profesores y estudiantes de todas las ramas del procesamiento del lenguaje natural. Actualmente el sistema se encuentra en su versión 5.0 y cuenta con un equipo de alrededor de veinte desarrolladores activos. Se encarga de muchas y muy diversas tareas como son: anotación manual, evaluaciones de rendimiento, extracción de información, anotación semántica semi-automática, etc. La distribución estándar cuenta con cincuenta *plugins*, lo cual hace una idea de la magnitud de la herramienta.

Los creadores en (The University of Sheffield), indican que GATE es altamente eficiente: en su última versión mejora un 20% la velocidad y un 40% el consumo de memoria y contiene un motor de procesamiento de texto basado en maquinas de estados finitos altamente eficiente. A su vez dispone de muchos plugins con tiempo de ejecución lineal. También, los responsables comentan de su producto que es calificado como “sobresaliente” y “líder internacional” por una evaluación anónima de la EPSRC.

Esta arquitectura permite por su diseño (véase Ilustración 2.2) utilizar multitud de tipos de datos para la entrada de información: XML, HTML, PDF, MS Word, correo electrónico, texto plano, etc. En cuanto al almacenamiento de datos, GATE proporciona una capa de almacenamiento persistente que da soporte a tecnologías como XML, Oracle, Postgre SQL y serialización Java. También permite la interoperación de entrada/salida con muchos otros sistemas.

Este sistema está diseñado bajo una arquitectura modular, en la que cada componente contiene unas interfaces bien definidas. Los principales componentes de este sistema son:

- LanguageResources (LRs): representan lexicons, corpus, ontologías, etc.
- ProcessingResources (PRs): representa entidades que son principalmente algorítmicas, como son analizadores, generadores o modeladores de n-gramas.
- VisualResources (VRs): representan la visualización y la edición de los componentes anteriores. Pertenece a este conjunto las interfaces de usuario.

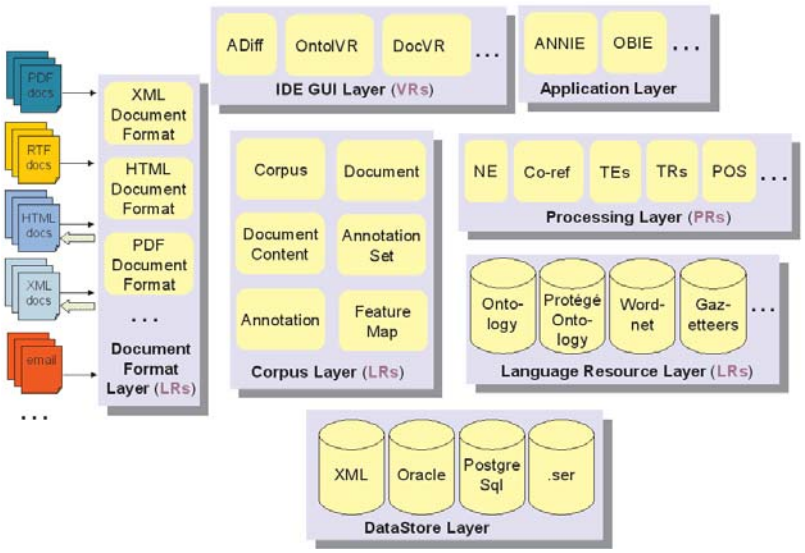


Ilustración 2.2: Arquitectura modular de GATE (The University of Sheffield)

Una arquitectura de este tipo para una herramienta de adquisición de información resulta atractiva ya que favorece en gran medida la investigación colaborativa, contiene una distinción muy bien definida entre tareas de bajo nivel (almacenamiento, visualización, etc.) y los componentes que desempeñan tareas de procesado de lenguaje de alto nivel. Define y limita limpiamente las estructuras de datos de los algoritmos de procesado de lenguaje natural.

2.2.2 Natural Language ToolKit (NLTK)

NLTK es un conjunto de programas y recursos en forma de librerías Phyton con el fin de aplicar diferentes técnicas lingüísticas de ámbito académico a colecciones de datos textuales. Incluye multitud de programas de ejemplo para las tareas más comunes en el procesamiento de lenguaje natural como la tokenización, etiquetación y anotación,

traducción, desambiguación, etc. Estas librerías las suelen utilizar estudiantes para facilitar la escritura de programas que realicen estas tareas. NLTK es gratuito bajo licencia GNU.

Los autores Edward Loper y Steven Bird introducen, en el primer artículo publicado sobre este conjunto de librerías(Edward Loper, 2002), que existen herramientas sobre el tratamiento de lenguaje natural en diversos lenguajes de programación ya que muchos de estos, sus estructuras y su lenguaje favorecen a un tipo de tratamiento, por ejemplo, se suele utilizar Prolog para realizar análisis gramaticales, Perl para procesamiento de Corpus y alguna herramienta de estados finitos para el análisis de la estructura morfosintáctica.

Este conjunto de herramientas esta implementado como una colección de módulos independientes, cada uno de los cuales define una estructura de datos o una tarea específica. Un conjunto de módulos centrales definen tipos básicos de datos y sistemas de procesamiento que son usados a lo largo de todo el toolkit. El modulo **token** proporciona clases básicas para el procesamiento de elementos individuales de texto como palabras o pequeñas frases. El módulo **tree** define estructuras de datos para representar estructuras arbóreas sobre un texto como árboles sintácticos o morfológicos. El módulo **probability** implementa clases que codifican distribuciones de frecuencia y probabilidad e incluye una gran variedad de técnicas de suavizado estadístico.

Uno de los principales módulos es el de análisis gramatical (**parsing module**) el cual define una interfaz de alto nivel para producir árboles que representen las estructuras textuales. Toda herramienta de procesamiento de lenguaje natural ha de representar de una u otra manera las estructuras del lenguaje.

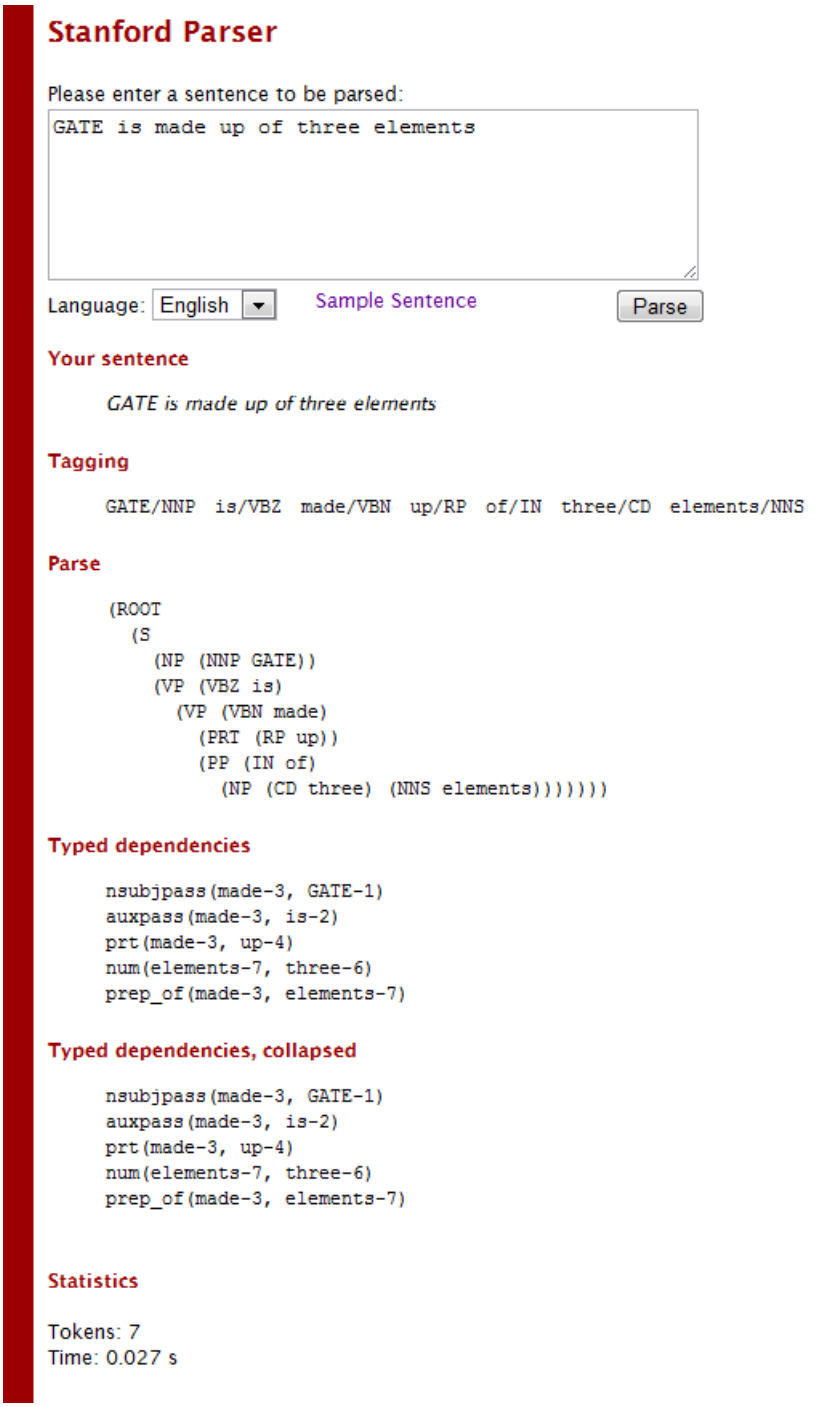
El apartado más relacionado de NLTK con respecto a COGNOS.NL se encuentra en el capítulo 10 (y más brevemente en el apartado 1.5) del libro Natural Language Processing with Python (Steven Bird) que aproxima estas herramientas para realizar ya no un análisis sintáctico sino un análisis del significado de las palabras. Esta herramienta proporciona una gran diversidad de medios para realizar esta tarea, desde la traducción de preguntas en lenguaje natural a consultas en lenguaje SQL a la introducción de reglas lógicas como representación del conocimiento de un dominio, y su posterior utilización.

2.2.3 Stanford Parser: Un parser estadístico.

Un *parser* de lenguaje natural es aquel que calcula la estructura gramatical de las frases, por ejemplo, qué grupo de palabras van juntas y qué palabras son el sujeto o los complementos del verbo. Los *parsers* probabilísticos obtienen el conocimiento adquirido a través de frases anotadas manualmente para así intentar producir el análisis más común para nuevas frases. Estos parsers estadísticos siguen cometiendo errores de análisis pero suelen realizar su trabajo considerablemente bien.

Las ideas técnicas básicas de cómo estos parsers funcionan están presentes en (Dan Klein, Fast Exact Inference with a Factored Model for Natural Language Parsing, 2003), y en (Dan Klein, Accurate Unlexicalized Parsing, 2003). En estos artículos, los autores presentan un modelo de generación de estructuras arbóreas del lenguaje natural en el cual las estructuras semánticas y sintácticas están diferenciadas en modelos separados. Indican también que al factorizar el modelo de esta forma, este admite un algoritmo A* extremadamente eficiente el cual permite una inferencia exacta y eficiente. El parser también proporciona como salida relaciones tipadas, también conocidas como relaciones gramaticales.

En la página web de este parser (The Stanford Natural Language Processing Group) está presente, además de la información del proyecto, una demo online en la cual se permite al usuario introducir una frase, seleccionar un idioma de entre tres: inglés, chino y árabe, y realizar el análisis. Este consta de información acerca de la gramática (apartado parse), la anotación (apartado tagging) y las dependencias entre los diferentes tokens (apartado typed dependences). Por último es necesario aclarar la extrema velocidad del algoritmo, ya que en este ejemplo, para analizar la frase “GATE is made up of three elements”, el sistema ha tardado 0.027 segundos.



Stanford Parser

Please enter a sentence to be parsed:

GATE is made up of three elements

Language: English Sample Sentence Parse

Your sentence

GATE is made up of three elements

Tagging

```
GATE/NNP  is/VBZ  made/VBN  up/RP  of/IN  three/CD  elements/NNS
```

Parse

```
(ROOT
  (S
    (NP (NNP GATE))
    (VP (VBZ is)
      (VP (VBN made)
        (PRT (RP up))
        (PP (IN of)
          (NP (CD three) (NNS elements)))))))
```

Typed dependencies

```
nsubjpass(made-3, GATE-1)
auxpass(made-3, is-2)
prt(made-3, up-4)
num(elements-7, three-6)
prep_of(made-3, elements-7)
```

Typed dependencies, collapsed

```
nsubjpass(made-3, GATE-1)
auxpass(made-3, is-2)
prt(made-3, up-4)
num(elements-7, three-6)
prep_of(made-3, elements-7)
```

Statistics

Tokens: 7
Time: 0.027 s

Ilustración 2.3: Stanford parser online demo

3 Especificación de requisitos

En este apartado se describirán los requisitos del software a un nivel de detalle suficiente para diseñar un sistema que los satisfaga por completo. Tal y como detalla el estándar IEEE Std 830-1998 (Software Engineering Standards Comitee of the IEEE Computer Society, 1998), una buena especificación de requisitos ha de ser:

- Correcta. Cada requisito declarado debe encontrarse en el software reflejando las necesidades reales del usuario.
- Inequívoca. Cada requisito declarado tiene solo una interpretación, para ello se evitará usar palabras o descripciones ambiguas que puedan dar lugar a confusión.
- Completa. En esta especificación estarán presentes todos los requisitos relacionados con la funcionalidad que el sistema va a proporcionar.
- Consistente. Ninguno de los requisitos o conjunto de ellos presentes en el documento ha de generar conflicto con otros.
- Comprobable. Cada requisito ha de ser fácilmente verificable, esto significa que para cada requisito será posible realizar una prueba que lo valide.
- Modificable. La estructura y estilo de la especificación de requisitos facilitarán que se pueda realizar cualquier cambio en los requisitos de forma consistente.

A parte de la especificación de requisitos, en este capítulo se estudiará la viabilidad del proyecto, en el cual se realizará un análisis de las características del sistema con vistas a verificar si éstas se pueden cubrir. A su vez, en el último punto, se especificarán pruebas de verificación sobre los requisitos que permitirán comprobar si el producto final los satisface.

Especificación de requisitos software

En este apartado se enumeran y se explican brevemente los requisitos extraídos durante las diferentes reuniones con el usuario. Estos se dividen en requisitos generales y en específicos. Los primeros son los que detallan el propósito de la herramienta así como los requisitos de interfaces ya sea con el usuario o con otros sistemas. Los segundos describen más a fondo las capacidades que debe de tener la herramienta.

3.1.1 Requisitos generales

Estos requisitos son los requisitos que exponen el propósito general de la aplicación: para qué funcionalidades está enfocada y qué se desea de ella. Además estos requisitos sirven como introducción a los requisitos específicos. Este proyecto tiene como base tres requisitos principales:

Requisito 1: Se diseñará una base de conocimiento implementada sobre un sistema gestor de bases de datos.

Requisito 2: Se desarrollará una herramienta capaz de auxiliar en el análisis de lenguaje natural, almacenando los datos obtenidos en una base de conocimiento.

Requisito 3: Se desarrollará un motor de inferencia que sea capaz de extraer el conocimiento de la base de datos y con éste, interpretar expresiones de lenguaje natural.

3.1.1.1 Interfaces hardware

Requisito 4: Para ejecutar la esta herramienta se debe de tener una máquina que disponga como mínimo del siguiente hardware: procesador Intel™ Pentium ® 4 con 1 Gb de memoria ram y la tarjeta gráfica que soporte una resolución mínima de soportada mayor a 1014 de alto por 883 de alto.

Requisito 5: Para almacenar la base de conocimiento se requiere una maquina servidora con las siguientes características: 2 x Intel Pentium III con un almacenamiento físico de al menos 36 GB. Y 1 GB de memoria ram.

Requisito 6: El sistema recibirá la información por parte del usuario mediante un teclado y un ratón. La información de salida se mostrará al usuario a través de una pantalla.

3.1.1.2 Interfaces software

Requisito 7: La aplicación funcionará sobre los sistemas operativos Windows 2000, Windows XP y Windows Vista.

Requisito 8: Es necesario que el equipo tenga instalada la máquina virtual de Java versión 1.6 y posteriores.

Requisito 9: El Sistema Gestor de Base de Datos que almacenará la base de conocimiento será Oracle® 10g y deberá de estar instalado en la maquina servidora.

Requisito 10: Para permitir la ejecución de operaciones sobre la base de datos desde el lenguaje de programación Java es necesario que el equipo soporte el driver de conexión JDBC

3.1.1.3 Interfaces de comunicaciones

Requisito 11: Se requiere que el equipo en el que se ejecuta la aplicación se conecte al servidor de bases de datos mediante una red de área local utilizando para ello el protocolo TCP/IP.

3.1.1.4 Interfaces con el usuario

Requisito 12: La herramienta contará con una interfaz gráfica ágil, intuitiva y amigable debido a que los usuarios a los que está dirigida la aplicación son en su mayor parte lingüistas y por tanto es posible que no estén familiarizados con la informática.

3.1.1.5 Requisitos de seguridad

Requisito 13: El acceso al sistema se realizará a través de nombre de usuario y contraseña. Se utilizará el propio sistema de autenticación que proporciona el sistema gestor de base de datos.

3.1.2 Requisitos específicos

Requisito 14: El sistema debe de mostrar al usuario la expresión que será editada.

Requisito 15: El usuario debe de ser capaz de seleccionar porciones de expresión. La unidad mínima de estos fragmentos será la palabra.

Requisito 16: El usuario debe de poder editar patrones, sub-patrones y tokens. Un patrón se debe de poder descomponer en patrones, tokens (opcionales o no), variables (opcionales o no) y comodines.

Requisito 17: El usuario debe de poder asignar instancias de actos comunicativos a cada patrón.

Requisito 18: Un token se debe de poder descomponer en tokens (opcionales o no), parámetros (opcionales o no), comodines y palabras, las cuales serán los terminales de las gramáticas.

Requisito 19: Se debe de permitir al usuario especificar la cardinalidad máxima y mínima de los comodines así como definir como cardinalidad máxima infinito.

Requisito 20: El usuario podrá en cualquier momento reiniciar la edición del elemento en curso.

Requisito 21: El sistema deberá de mostrar los actos comunicativos que se asignan a cada patrón así como definir el orden dentro de la asignación y eliminarlos de ella.

Requisito 22: Para permitir al usuario seleccionar una instancia de un acto comunicativo, el sistema deberá de permitir la visualización, por cada acto comunicativo perteneciente al dominio (editados mediante Cognos.CA), de sus categorías, y por cada categoría, permitir la selección de uno de sus valores posibles.

Requisito 23: Respecto a la selección de actos comunicativos, el sistema deberá de filtrar los valores posibles por cada valor seleccionado anterior.

Requisito 24: La aplicación debe de permitir al usuario la verificación de la validez de la edición en curso.

Requisito 25: Siempre que una edición sea válida, el sistema debe de permitir al usuario insertar el conocimiento editado en la base de datos. Ya sea la edición de un patrón completo o un token completo.

Requisito 26: El sistema debe de ser capaz de inferir una propuesta de acto(s) comunicativo(s) para una expresión a partir del conocimiento editado previamente.

Requisito 27: El usuario podrá editar una frase que provenga de otras aplicaciones como Cognos.Dial, escribir una expresión manualmente ó importar una expresión proveniente de un fichero de texto.

Requisito 28: Para que la edición de un patrón sea válida, toda variable seleccionada ha de estar asociada al menos a una instancia de acto comunicativo, pudiendo aparecer en distintas categorías de esa instancia o en diferentes instancias asignadas.

Requisito 29: Para que la edición de un patrón sea válida, ha de tener al menos una instancia de acto comunicativo asignada.

Requisito 30: En la edición de tokens, el sistema debe de mostrar su colección de tokens editados previamente en cualquier dominio de interacción. La edición de tokens consta en insertar nuevos elementos a esta colección.

Requisito 31: Cada token consta a su vez de alternativas llamadas descripciones. El sistema permitirá al usuario elegir si se desea insertar una nueva descripción dentro de un token existente, reutilizar una descripción concreta ó insertar un nuevo token.

Requisito 32: Se debe de permitir asignar un dominio a los elementos editados como variables. Estos dominios podrán ser los siguientes: cantidad, fecha u hora.

Requisito 33: Se permitirá asignar o desasignar una variable a una categoría de una acto comunicativo asociada a la edición.

Estudio de viabilidad

En este apartado se realizará el análisis de las necesidades propuestas en el apartado anterior y se plantearán soluciones que las satisfagan teniendo en cuenta restricciones económicas, técnicas y operativas. Una vez descritas cada una de las alternativas se valoraran en cada caso las fortalezas y debilidades que ofrecen con el fin de seleccionar la más adecuada

3.2.1 Viabilidad del proyecto

Los usuarios a los que está enfocado el proyecto son en su mayor parte lingüistas, expertos en el análisis de lenguaje natural, que analizarán las expresiones para así alimentar el apartado de lenguaje natural de la base de conocimiento Cognos. Estos usuarios posiblemente no estén familiarizados con aplicaciones y entornos de edición complejos y con muchas opciones, por lo cual es necesario especificar un sistema fácil de usar y de aprendizaje ágil. Para ello la interfaz de usuario (UI) deberá ser todo lo intuitiva y amigable posible.

Para la creación de una interfaz como la comentada en el párrafo anterior es necesario un lenguaje con librerías y API's que dispongan de buenas posibilidades gráficas. Un lenguaje orientado a objetos facilitará la encapsulación, modularidad y en definitiva un buen diseño de la aplicación. Por tanto se utilizará el lenguaje de programación Java, que satisface estas necesidades, además es un lenguaje pre-

compilado por lo que el tiempo de respuesta será menor, favoreciendo de esta forma la experiencia de usuario.

El sistema estará basado en una arquitectura cliente-servidor por lo cual las especificaciones tecnológicas se dispondrán siguiendo dos enfoques: el enfoque del cliente y el del servidor. Es necesaria esta división ya que los requisitos tanto de hardware como de software para cada uno de ellos divergen bastante.

3.2.1.1 Cliente

Por un lado será necesaria la máquina que actúa como cliente la cual ejecutará la aplicación y permitirá realizar el análisis de expresiones. Al estar esta aplicación programada en Java, la máquina debe de ser capaz de ejecutarla, por tanto se ha de disponer al menos del entorno de ejecución o máquina virtual de Java. Se recomienda al menos disponer del entorno de desarrollo Java en alguna de las máquinas clientes.

Como sistema operativo para el cliente será necesario un sistema operativo que ofrezca soporte actual y que disponga de la mayor difusión posible. Las diferentes opciones actuales son utilizar Windows o utilizar Linux. A favor del primero, está la amplia difusión con la que cuenta y que prácticamente la totalidad de usuarios de ordenadores personales lo utiliza. En su contra corre que es un sistema de pago. El sistema operativo podrá ser tanto de 32 como de 64 bits.

Debido al tipo de operaciones que efectúa el motor de inferencia, la carga de datos a memoria principal es más viable; por un lado ya que la velocidad de la memoria principal es inmensamente superior y no depende en absoluto de otros factores como la latencia de red, y por otro, las operaciones comentadas son en base recursiva, por tanto, es común tener que consultar varias veces un valor. Si este no se almacena en memoria principal, el usuario no apreciará un mecanismo ágil de sugerencia y por otro lado, se sobrecargará la red.

Por lo dicho anteriormente, en cuanto a especificaciones hardware para la parte del cliente deberá de ser de una máquina con memoria RAM suficiente como para permitir la carga del conocimiento almacenado para así permitir al motor de inferencia realizar la sugerencia en base a este.

En cuanto a capacidad de procesamiento, la maquina cliente no ha de ser más potente que cualquier ordenador personal común hoy en día. Ya que se trata en su

mayor parte de una herramienta de edición, las operaciones de cálculo se reducen a la inferencia en la cual la velocidad de cálculo no es tan determinante como es la de acceso a los datos.

3.2.1.2 Servidor

Esta herramienta, necesita almacenar los resultados de la edición en una base de conocimiento implementada en un sistema gestor de base de datos (S.G.B.D.). Este ha de ser concurrente ya que múltiples clientes realizarán anotaciones al mismo tiempo. También se busca facilidad en la implementación de la comunicación entre la base de datos con la herramienta. Existe la posibilidad de prescindir de un sistema gestor de base de datos e implementar el almacenamiento en ficheros XML, pero esto complicaría de sobremanera la implementación de la aplicación. Se tomará por tanto Oracle 11g como sistema gestor de bases de datos para este proyecto ya que satisface las necesidades.

En cuanto al sistema operativo (S.O.) de la máquina servidora, se utilizará un sistema operativo un poco más potente que el de la máquina cliente y deberá ser un S.O. específico para servidores, ya que estos ofrecen una mejora notable en lo que a comunicaciones y configuraciones se refiere. Unas posibles alternativas actuales serían utilizar Windows Server (2003 o 2008) o Alguna distribución Linux como Ubuntu Server.

Con respecto al hardware del servidor ha de ser bastante más potente que el de las máquinas clientes, ya que es el que se va a encargar de recibir la información concurrente por parte de estas. Por lo cual sería recomendable disponer de una máquina que disponga de más de un procesador, para así poder hacer frente a cualquier número de peticiones de servicio que reciba.

En cuanto a memoria RAM se refiere, sería recomendable disponer de la cantidad que actualmente ofrecen los servidores en el mercado, pero con un margen considerable, por tanto, esta cantidad podría oscilar en torno a 4 – 16 Gb de memoria RAM.

3.2.2 Especificaciones técnicas

Como se ha especificado en el apartado anterior, ningún requisito respecto a tecnología es inviable, y todos ofrecen un rango bastante amplio de decisión en cuanto a potencia, rendimiento y presupuesto. En este apartado se detallarán las decisiones

tomadas en cuanto a especificaciones técnicas para comenzar con la realización del sistema.

3.2.2.1 Cliente

Como se ha comentado, cualquier ordenador comercial cumple con creces los requisitos técnicos para la parte del cliente, se exponen los requisitos mínimos orientativos.

Configuración mínima:

- Sistema Operativo: Microsoft Windows XP Professional.
- Procesador: AMD Athlon 64, 2000 MHz (10 x 200) 3000+, ó correspondiente Intel.
- Memoria RAM: 512 MB.
- Almacenamiento: Disk Device (80 GB, 7200 RPM, SATA).

3.2.2.2 Servidor

La máquina servidora debe de disponer de una configuración típica para ordenadores que desempeñen este tipo de labores, son especificaciones más restrictivas que la máquina cliente.

Se propondrán dos configuraciones, mínima y recomendada, ambas completamente viables actualmente ya que se corresponden con las dos máquinas que el grupo LABDA pone a disposición de este proyecto. La configuración mínima pertenece a un equipo con más de cinco años de antigüedad y la configuración recomendada pertenece a un servidor con unos tres años de antigüedad, por tanto cualquier servidor comercial actual podrá satisfacer los requisitos técnicos.

Configuración mínima:

- Sistema Operativo: Microsoft Windows Server 2003, Enterprise Edition.
- Procesador: 2x Intel Pentium IIIE, 1000 MHz (7.5 x 133).
- Memoria RAM: 1024 MB.
- Almacenamiento: SCSI Disk Device (36 GB, 10000 RPM, Ultra320 SCSI).

Configuración recomendada:

- Sistema Operativo: Microsoft Windows Server 2003, Enterprise Edition. o superior.
- Procesador: AMD Opteron Quad-Core Processors: 2356 (2.3 GHz, 75W ACP), 2354 (2.2 GHz, 75W ACP), 2384 (2.7 GHz, 75W ACP), 2376 (2.3 GHz, 75W ACP).
- Memoria RAM: 4096 MB.
- Almacenamiento: 146 G 15K rpm 3.5" SAS drive.

Pruebas definidas sobre los requisitos

En este apartado se detallaran las pruebas de validación necesarias para comprobar que el sistema cumple satisfactoriamente los requisitos detallados en el apartado. Estas pruebas son diseñadas tras la descripción de requisitos y son necesarias para verificar que el sistema se ha desarrollado atendiendo a todas las necesidades.

Los requisitos del apartado 3.1.1 no son validados ya que definen los requisitos generales que siempre ha de satisfacer la herramienta, aparte de algunas especificaciones que tampoco han de ser validadas. Por tanto las pruebas siguientes están definidas sobre los requisitos presentados en el apartado 3.1.2.

Cada prueba atenderá a la validez de un requisito en concreto, pero es posible que una prueba valide más de uno, esto ocurre ya que algunos requisitos tienen referencias directas con otros, esto se podrá comprobar en la matriz de correspondencia requisito - prueba que se incluye posteriormente.

3.3.1 Definición de pruebas

Prueba 1: Tras arrancar la aplicación, insertar texto manualmente como expresión en el campo correspondiente, después de esto, seleccionar un fragmento de dicho texto.

Prueba 2: Comprobar que es imposible realizar cualquier selección de palabras a medias.

Prueba 3: Arrancar la aplicación, seleccionar una porción de texto y editarla como token, seleccionar otro fragmento y editarlo como patrón, seleccionar otro fragmento y editarlo como variable y realizar la misma operación editando el

fragmento como comodín. Se verificará que todos los fragmentos corresponden con el tipo de edición que se ha dispuesto para cada uno de ellos.

Prueba 4: Comprobar que existe un mecanismo de visualización y selección de actos comunicativos. Una vez seleccionado un tipo de acto comunicativo verificar que se muestran sus categorías y al seleccionar un valor de cada categoría, comprobar que los valores de las categorías siguientes se modifican para mostrar los valores válidos para la selección anterior.

Prueba 5: Mientras se está editando un patrón, verificar que es posible seleccionar un acto comunicativo de los editados mediante la herramienta Cognos.CA y es posible asignarlo al patrón del cual se está realizando la edición.

Prueba 6: Verificar que en la edición de un token, se pueda seleccionar fragmentos de expresión y estos puedan ser etiquetados como token, variable y comodín.

Prueba 7: Se comprobará que en la edición de un comodín es posible especificar el número de palabras máximo y mínimo que pueden aparecer en esa posición. Para ello se selecciona y se edita una o varias palabras como comodín y se verificará que aparece marcado con su cardinalidad editada.

Prueba 8: Mientras se esté editando un token, verificar que existe un mecanismo para poder limpiar la edición en curso y partir de cero.

Prueba 9: Tras haber realizado cualquier número de asignaciones de instancias de actos comunicativos al patrón en curso, verificar que éstos son visibles y existe un método que permita modificar el orden de dicha colección, así como algún medio para eliminar una instancia de acto comunicativo seleccionada.

Prueba 10: Después de haber hecho un análisis completo comprobar la validez de la edición, se verificará que la salida de dicha verificación es congruente para un patrón bien editado y para un patrón mal editado.

Prueba 11: Comprobar que la inserción del patrón se ha realizado con éxito en la base de datos mediante algún sistema externo de administración de base de datos como SQL developer.

Prueba 12: Verificar que existe un mecanismo de proposición de instancias de actos comunicativos para la expresión inicial. Se comprobará a su vez que dicho acto o actos comunicativos propuestos son coherentes con la expresión.

Prueba 13: Externamente se editará una muestra perteneciente a un Corpus la cual contendrá intervenciones. comprobar que dicha intervención es accesible mediante la interfaz de Cognos.NL y es posible editar un patrón asociado a esta.

Prueba 14: Corroborar que la acción de validar la edición no es correcta si se han editado variables y éstas no han sido asignadas a ninguna categoría de cualquier instancia asociada a la edición. Por el contrario, si todas las variables han sido asignadas a instancias de actos comunicativos, se comprobará que el sistema da como válida la edición del patrón.

Prueba 15: Comprobar que un patrón no es válido si no posee al menos una instancia de acto comunicativo asociada.

Prueba 16: Verificar que si se edita un fragmento de expresión como token, en la interfaz de edición de token aparecen todos los nombres de tokens editados previamente. Cotejar los datos de la interfaz con los existentes en la base de datos mediante alguna herramienta de gestión de base de datos como SQL Developer.

Prueba 17: Comprobar que si se marca la opción de reutilizar una descripción existente, se asignará al patrón en edición esa descripción, por lo cual en la base de datos no aparecerá de nuevo otra descripción.

Prueba 18: Verificar: tras haber asignado un acto comunicativo y una vez editado un elemento como variable, es posible seleccionar una categoría del acto comunicativo en cuestión y asociarle la variable en vez de su valor fijo. Esto se mostrará si el valor de dicha categoría es modificado en la lista de actos comunicativos.

3.3.2 Matriz de trazabilidad

En esta tabla se mostrará la correspondencia que tienen las pruebas con respecto a los requisitos que pretenden validar. También es útil para ilustrar cuándo una prueba valida más de un requisito.

La prueba número siete aparece en esta matriz como que realiza la validación de dos requisitos, pero esto es debido a la dependencia entre estos. Para realizar la edición de un comodín e insertar sus parámetros (cardinalidades), es necesario que se haya seleccionado una porción de expresión y editada como comodín, pero la prueba con la que que compite, es decir, la prueba 3, verifica la totalidad de opciones de edición.

Requisito-Prueba	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17	P19
R14	X																	
R15		X																
R16			X				X											
R17					X													
R18						X												
R19							X											
R20								X										
R21									X									
R22				X														
R23				X														
R24										X								
R25											X							
R26												X						
R27													X					
R28														X				
R29															X			
R30																X		
R31																	X	
R32																		X
R33																		X

Tabla 1: Matriz prueba-requisito

Metodología de desarrollo

El ciclo de vida que seguirá el proyecto es el modelo en espiral, es un modelo que combina las características de los modelos en cascada y de prototipos. El ciclo de vida en espiral está pensado para proyectos largos, caros y complicados en el cual la línea base transcurre a través de sus etapas varias veces a lo largo de su vida. En la Ilustración 3.1 se muestra gráficamente un ejemplo de este tipo de ciclo de vida, el cual conlleva las siguientes fases en cada una de sus iteraciones.

Planificación: Fase en la cual se determinan los objetivos del proyecto, se identifican los requisitos. Esta definición de requisitos se irá enriqueciendo cada vez que la línea base transcurra a través de esta fase añadiendo, eliminando o modificando requisitos como a su vez modificando su prioridad.

Análisis de riesgos: En esta fase se identifican y se resuelven los riesgos mediante un estudio de viabilidad de la iteración. El producto esta fase por cada iteración es un prototipo viable con la menor cantidad de riesgos posible.

Desarrollo: Es en esta fase en la cual se realizan la codificación y pruebas de cada prototipo, dando como resultado un producto por iteración. A lo largo de esta fase se pueden englobar tareas tales como la realización de un diseño detallado o la integración del sistema.

Evaluación: En esta fase se evalúa el producto de la fase anterior, evaluación en la cual se extraen las deficiencias de este, las cuales son empleadas para realizar la planificación de la siguiente fase que será de nuevo la planificación hasta que el sistema llegue al punto de lanzamiento.

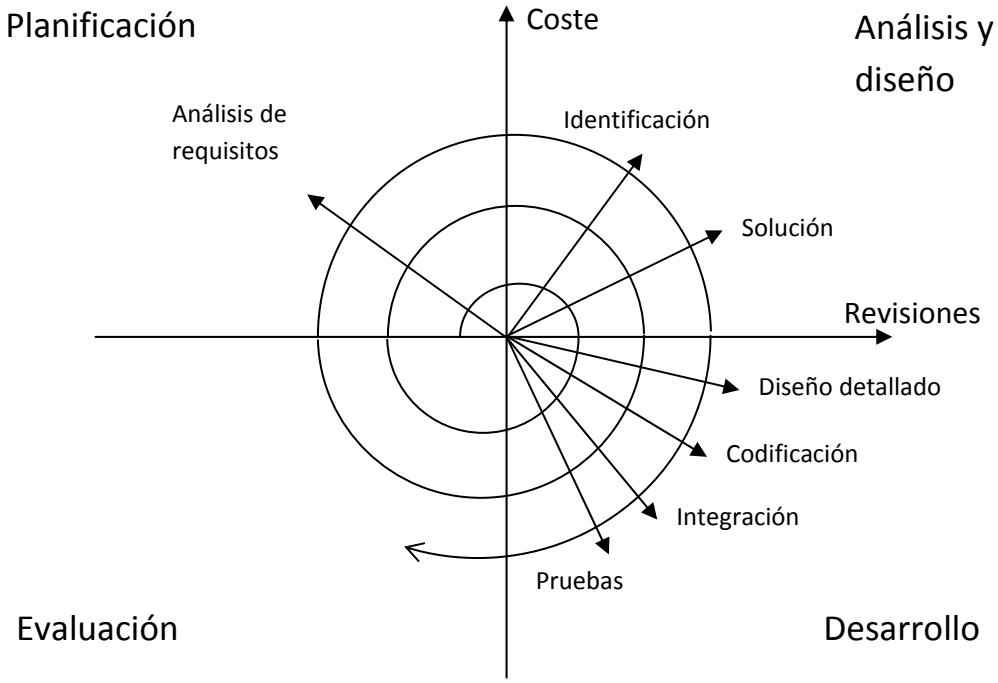


Ilustración 3.1: Ciclo de vida en espiral

Este proyecto inicialmente se planteó como una ampliación de la herramienta de anotación de corpus de diálogo COGNOS.Dial, pero finalmente se decidió por una herramienta independiente pero integrada en el mismo toolkit. En la primera iteración se propone una maqueta gráfica de la herramienta sin profundizar en el aspecto funcional de la misma: se adquieren los requisitos de ésta, se diseña, se implementa y, una vez evaluada, se aumentan los requisitos (siguiente iteración), esta vez dando prioridad a los requisitos funcionales, para otorgar al esqueleto implementado de ciertas operaciones que permitan empezar a satisfacer requisitos generales del sistema. En iteraciones sucesivas se irá ampliando requisitos, sobretodo de aspecto funcional.

4 Análisis y diseño

Este apartado se expondrá el planteamiento del sistema a un nivel superior de abstracción, sin llegar a especificar un diseño detallado se describirá la relación entre los requisitos funcionales y las soluciones que se han adoptado para darles soporte. En un primer nivel se comentará un diseño conceptual del sistema, en el cual se expondrán las características del diseño del sistema así como los distintos módulos que ofrecerán la funcionalidad requerida. Por otro lado se especificarán las necesidades de información del sistema, es decir, qué información maneja y opera el sistema, dónde y cómo se almacena, se recupera y se utiliza para poder desempeñar toda la funcionalidad. Por último se detallarán los prototipos de interfaces de usuario diseñadas, su flujo y consecución.

Arquitectura física del sistema

Como se ha mencionado el sistema se basa en una arquitectura cliente-servidor en la cual las aplicaciones clientes editan las expresiones y el conocimiento se almacena en una base de datos común presente en el servidor. El servidor permanecerá activado permanentemente para que siempre que un cliente necesite realizar anotaciones, disponga tanto del conocimiento previo como la posibilidad de insertar nuevo, sin necesidad de realizar un arranque de dos máquinas.

Una arquitectura de este tipo mejora considerablemente la velocidad de adquisición de conocimiento ya que múltiples usuarios pueden analizar todas las expresiones de un corpus en paralelo. Otra posibilidad sería utilizar este paralelismo para verificar y validar dos anotaciones de la misma expresión y compararlas, para, de este modo, obtener un conocimiento más pulido.

En la Ilustración 4.1 se aprecia el servidor, que es el que posee la base de conocimiento y por otro lado las diferentes aplicaciones que actúan como cliente,

realizando inserciones y consultas en esta base de datos. Para que esto sea posible, se requiere que todas las máquinas que ejecutan el cliente, dispongan de conexión a internet y acceso a la máquina servidora.

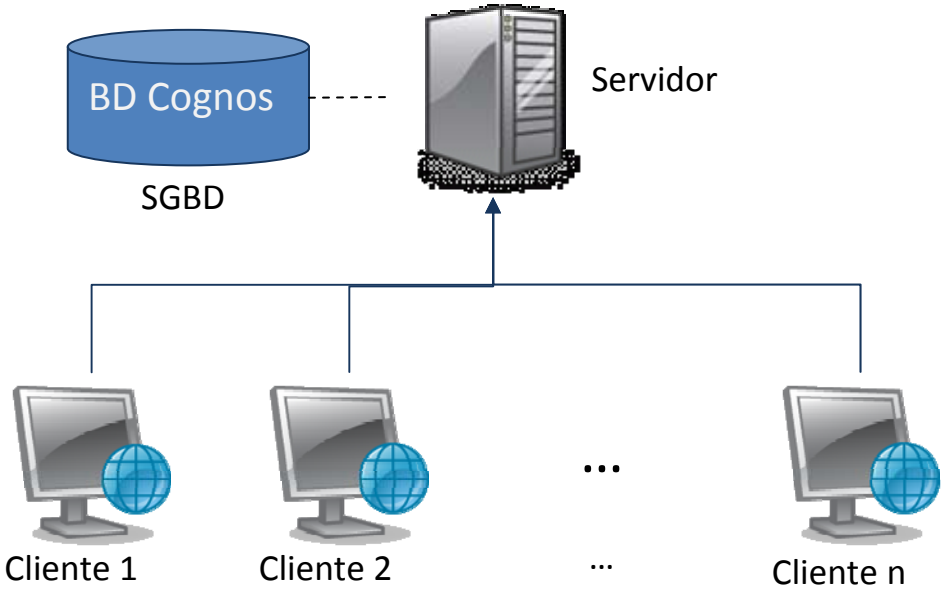


Ilustración 4.1: arquitectura cliente-servidor

La anotación de expresiones de lenguaje natural provenientes del corpus para poblar una base de conocimiento es una tarea que conlleva una ingente cantidad de trabajo. De ahí la necesidad de que existan varios expertos trabajando en ello. Si se diseñase la arquitectura del sistema como un conjunto de aplicaciones totalmente independientes, con su propia base de conocimiento, sería necesario unificar todas las bases de datos individuales en una global en caso de que sean anotaciones en caso de que el trabajo se distribuyese entre los diferentes usuarios. Pero si se desease realizar un trabajo paralelo de las mismas expresiones para realizar una evaluación, este proceso se complicaría de sobremanera. Queda justificada pues la decisión de llevar a cabo una arquitectura cliente-servidor.

Arquitectura funcional

El objetivo principal del sistema es obtener una herramienta capaz de permitir al usuario editar y almacenar de forma ágil y sencilla gramáticas de lenguaje natural basadas en expresiones provenientes o no de un Corpus existente. Así como el almacenamiento de estas gramáticas en una base de conocimiento para que de esta forma, un procesador de lenguaje natural pueda inferir sobre ellas.

En este apartado se especifica el diseño a alto nivel de la aplicación. Para aportar la funcionalidad pedida se ha decidido plantear un diseño basado en el diseño "modelo-vista-controlador" pero contando con una capa que ofrece la funcionalidad de las inferencias en base al conocimiento. Más adelante se explicará esta funcionalidad.

En la Ilustración 4.2 se puede apreciar el diagrama de componentes el cual presenta los principales paquetes: Modelo, Vista y Controlador. Cada uno de ellos tiene unas funcionalidades definidas y requieren de otros para completar sus tareas. Se detallará a continuación todos y cada uno de los paquetes y componentes presentes en la ilustración.

El diseño arquitectónico se ha realizado tomando como base el del patrón modelo-vista-controlador, pero se usó como referencia, sin llegar a implementar su base a fondo, ya que el comportamiento dotado para cada uno de los componentes no responde exactamente con el esperado para este estándar arquitectónico. De ahí el nombre de los principales paquetes. Se irán viendo las diferencias en los apartados siguientes.

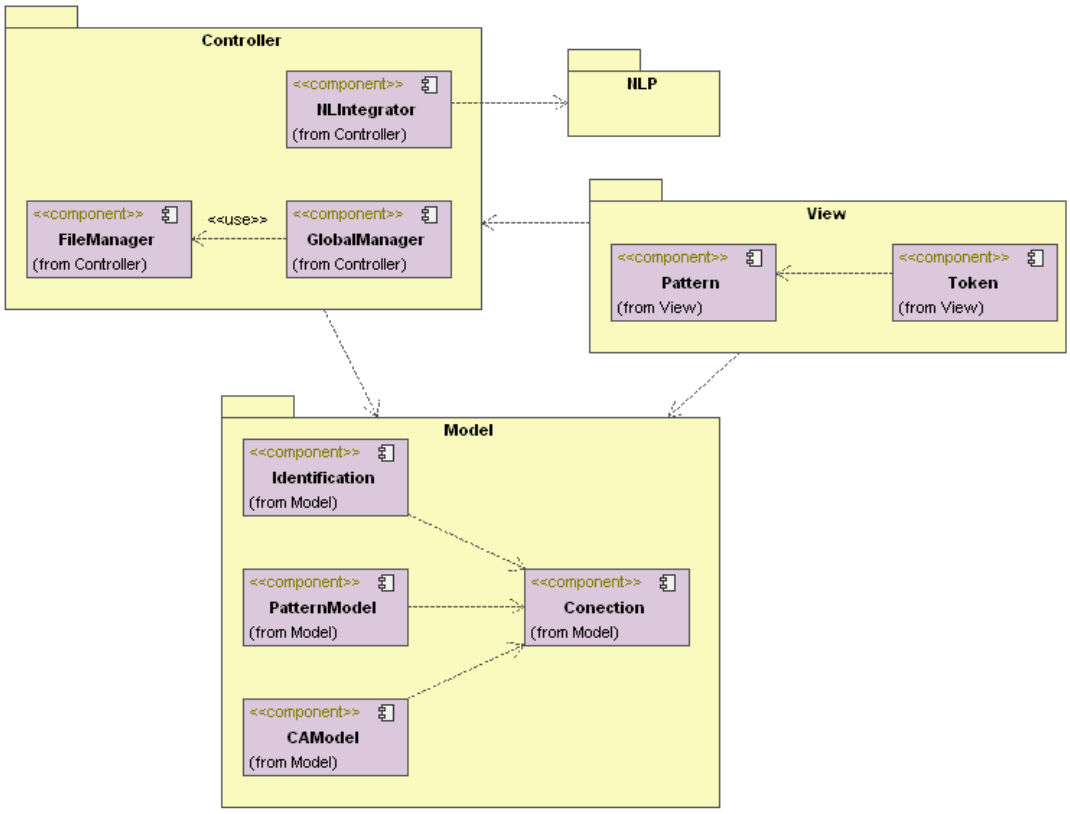


Ilustración 4.2: Diagrama de componentes

4.2.1 Vista (View)

El paquete vista engloba todos los aspectos relacionados con la interfaz de usuario, por tanto, es el medio de comunicación de la herramienta con este. Al ser el proyecto una herramienta para facilitar al usuario la anotación, el componente de vista recibe la información en forma de acciones realizadas sobre la interfaz, e informa al usuario en todo momento del estado de las acciones mediante diversas técnicas:

- **Color de la selección del elemento.** Se especificará un código de colores para cada uno de los elementos editables. De esta forma el usuario reconocerá rápidamente el tipo de elementos editados. Debido a que la edición de sucesivas expresiones es una tarea mecánica y muy repetitiva, los colores favorecen a los usuarios avanzados a agilizar el análisis, pero se sacrifica tiempo de entrenamiento para nuevos usuarios, que deberán de conocer el abanico de colores para así poder sacar beneficio de ellos.

- **Ventanas emergentes.** El sistema dispondrá de un sistema de ventanas emergentes o *pop-ups*. Las cuales se activaran principalmente en dos casos: como advertencia al usuario de que la acción que va a emprender puede desencadenar pérdida de información ó como selector de datos, en las cuales se ofrecerá al usuario distintas posibilidades de elección para facilitar el relleno de formularios.

- **Panel persistente de información.** El sistema contará con un panel informativo en el cual siempre se mostrará si una acción ha sido realizada correctamente, si ha habido un error y no se ha podido realizar algo, etc. Por tanto en este texto siempre aparecerá la última acción realizada hasta que se realice otra.

Como se explicó anteriormente, una expresión es por sí sola un patrón, pero puede contener sub-patrones o tokens. Estos últimos poseen como principales diferencias que no contienen actos comunicativos asociados y que no pueden ser compuestos por más sub-patrones. Estas pequeñas distinciones conllevan a que ambas ediciones tengan que disponer de interfaces completamente distintas.

En común disponen de varios controles como son la inserción de variables, comodines y otros tokens, pero los patrones pueden disponer de sub-patrones y los

tokens pueden tener los terminales de la gramática correspondiente a la expresión, los cuales son las palabras literales que la componen. La edición de una expresión por lo comentado anteriormente podría ser en mayor o menor medida realizada en forma arbórea.

La edición de variables conlleva una diferencia entre ambos tipos de ediciones. Es imprescindible para una edición completa y válida de una expresión que todas y cada una de las variables sean asignadas a categorías de los actos comunicativos seleccionados para la expresión. Cuando se edita un token, al este no tener asociados actos comunicativos, las variables serán tomadas por la primera edición de patrón o sub-patrón y en la edición de éste será en la cual se deba de asignar las variables a los actos comunicativos. Esto se ha realizado así para dotar de mayor potencia a la herramienta de edición ya que para que al editar un token pesado se pueda subdividir en varios tokens más pequeños sin perder consistencia y validez.

4.2.1.1 Vista.Patrón (View.Pattern)

Este componente es el encargado de ofrecer la funcionalidad para las ediciones de patrones y sub-patrones. Para ello dispondrá de mecanismos por los cuales el usuario será capaz de seleccionar porciones de la expresión y seleccionar de qué tipo de elemento se trata. Cada elemento seleccionado tendrá unas opciones de edición distintas. Los sub-patrones y los tokens lanzan instancias de nuevas interfaces para ser anotados independientemente. A las variables se les asignará un nombre y posteriormente se editará su información y por último los comodines dispondrán de datos sobre sus cardinalidades máxima y mínima así como su posibilidad de ser infinitos.

La vista de la edición del patrón raíz o general posee unas funcionalidades añadidas a la vista de edición de sub-patrones, estas son:

1. Identificación de una expresión proveniente de un corpus ya presente en la base de datos (insertado y anotado mediante Cognos.Dial). para ello se deberá de seleccionar el corpus, el escenario, la muestra, la intervención y por último la expresión que se desea. Esto es necesario realizarlo así, porque de esta forma se realiza la identificación de expresiones esta parte de base de conocimiento COGNOS, como se verá más adelante. Para

esta función se dispondrá de 5 selectores en los cuales aparecerá la clave del elemento a seleccionar más una breve descripción.

2. Creación de nuevas expresiones. Se permitirá escribir frases en lenguaje natural en el mismo campo en el que aparecerían las expresiones identificadas. Se podrá analizar normalmente y el patrón resultante se almacenará en un conjunto de expresiones generales, este conjunto será utilizado siempre y para todos los Corpus. Es como un factor común del conocimiento de cada uno de ellos.

3. Acciones para guardar la edición en curso, limpiar la edición completamente, renombrar (almacenar en otro identificador) sin eliminar el original y borrar la edición en curso.

Como se ha comentado en apartados anteriores, un patrón, a diferencia de un token, ha de tener asociado uno o varios actos comunicativos para que aporte funcionalidad al modelo de diálogo. Por tanto las interfaces correspondientes a la edición del patrón raíz como de sub-patrones, han de disponer de un selector de actos comunicativos. Estos actos comunicativos estarán previamente editados genéricamente en la base de datos mediante la aplicación Cognos.CA. Una vez seleccionado la instancia de acto comunicativo genérico, se añade a la edición y se mostrará al usuario.

Como parte de la información hacia el usuario, el sistema mostrará los actos comunicativos asociados a los patrones hijos, los cuales únicamente se irán mostrando en los padres como información, ya que suponen un conocimiento implícito. En la siguiente ilustración se puede ver que el acto comunicativo *greet* que se encuentra en primera posición es de otro color, esto indica que proviene de un sub-patrón y es un elemento meramente informativo de los actos comunicativos de los que consta la expresión.

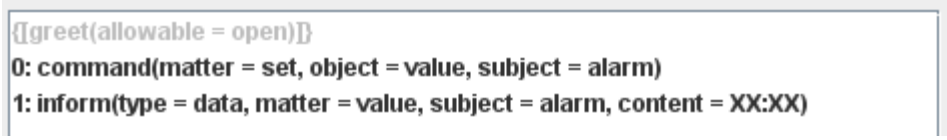


Ilustración 4.3: Actos comunicativos seleccionados.

Las variables editadas o heredadas por parte de sus elementos hijos han de ser asociadas a los actos comunicativos seleccionados por tanto se deberá de permitir

seleccionar cual categoría contiene la variable. También será necesario permitir editar una variable y asignarle un dominio. Los dominios que se permiten asociar serán:

- Fecha: una variable es de tipo fecha cuando es del tipo DD/MM/AAA. Se puede anotar como fecha cualquier cadena de caracteres, pero al asignar el dominio, se acepta este formato y la aplicación, internamente representa el conocimiento de las fechas de esta forma.

- Hora: una variable es de tipo hora cuando es del tipo HH:MM ó HH-MM. Sucede lo mismo que para las fechas, este formato es el soportado e insertado por la aplicación internamente, aunque se pueda anotar cualquier cadena de caracteres.

- Cantidad: una variable que responde al dominio de cantidad involucra cualquier cantidad de caracteres numéricos. Ocurre exactamente lo mismo que con hora y fecha.

Cuando se anota una variable, en vista de patrón, su valor en la expresión pierde su significado, tomando como tal el dominio seleccionado. Por tanto y por ejemplo, si se selecciona la palabra “dos” en una expresión y se anota como variable de dominio cantidad, en la base de conocimiento figurará que en ese punto del patrón será válido cualquier número de dígitos, pero no caracteres.

Por otro lado cuando el usuario desee inferir sobre los patrones almacenados previamente en la base de conocimiento, la interfaz, si la inferencia ha devuelto un resultado, es decir, uno o varios actos comunicativos, éstos se añadirán automáticamente a la selección que hubiera previamente.

4.2.1.2 Vista.Token (View.Token)

Este componente se encarga de ofrecer al usuario la funcionalidad de ediciones de tokens. Para ello, como la edición de patrones, permitirá al usuario visualizar la expresión o porción de expresión a editar. Cuando en la edición de patrón (o de token) se selecciona un fragmento de texto y se identifica como token, entra en funcionamiento este componente, el cual ofrecerá la posibilidad de editar la porción seleccionada anteriormente como un nuevo token, reutilizar el nombre y añadir una nueva edición o reutilizarlo completamente.

La interfaz de edición de tokens permite al usuario, como la análoga para patrones, de seleccionar porciones de texto de la expresión y marcarlas como otros tokens (opcionales o no, y por lo tanto, se instanciará una nueva interfaz para la porción seleccionada), variables (opcionales o no) y comodines con sus valores. Esta edición se podrá realizar siempre y cuando el usuario no desee reutilizar completamente un token.

Una descripción, como se explicó anteriormente, es una opción de gramática o palabra para un token, por ejemplo, el token "saludo" puede contener varias descripciones que en este caso serían las palabras "hola", "buenas". Cuando el usuario edite un nuevo token tiene la posibilidad de crear otro nuevo (totalmente independiente a los que existan previamente), añadir una descripción a un token previo o reutilizar completamente una descripción. Se analizarán estos casos uno a uno:

- Crear un nuevo token: para ello existirá un campo en el cual el usuario podrá especificar el nombre del nuevo patrón, se habilitarán los controles de edición y ese patrón será añadido a la base de datos como nuevo patrón con una primera descripción que será la editada para la expresión.

- Crear una descripción asociada a un token: para lo cual se mostrará una lista con los nombres de tokens correspondientes. Se deberá seleccionar uno de esta lista y editar la expresión. Cuando se seleccione un token a su vez se mostrarán en otros campos las descripciones que posee.

- Reutilizar totalmente una descripción: existirá un botón o mecanismo para permitir al seleccionar un token y una descripción, indicar al sistema que se desea reutilizar dicha descripción. Un ejemplo sencillo: al tener que analizar muchas expresiones, es muy posible que aparezca la palabra "hola", por tanto para no saturar la base de datos se reutilizará si existe la descripción "hola" del token que la contenga.

Un elemento fundamental de los tokens son las palabras (words). Éstas son los terminales de las gramáticas y por tanto toda formalización gramatical de una expresión será determinada por las palabras que la contienen. La interfaz no cuenta con un mecanismo implícito de marcado de palabras por tanto, se ha decidido que toda cadena restante en la edición de token será tomada como cadena de palabras literal. Por lo cual,

si se desease crear un token compuesto por un comodín y una palabra, bastará con marcar el comodín dejando libre el resto de expresión.

4.2.2 Modelo (Model)

Este paquete se encarga de las siguientes funciones:

- Acceso al almacenamiento. El almacenamiento es totalmente independiente a este paquete, pero en él se engloban las funcionalidades para permitir las conexiones, las inserciones y las adquisiciones de datos almacenados.

- Encapsulamiento de los datos. El modelo accederá a los datos almacenados y los encapsulará en objetos Java, los cuales serán accesibles por los distintos componentes que los requieran. Estos componentes se encargan de formular las sentencias SQL necesarias para obtener y representar los datos almacenados.

Por esto, se dispone de distintos componentes, los cuales son separados por funcionalidad y por tipos de datos encapsulados. A continuación se explicará cada uno de ellos.

4.2.2.1 Modelo.Conexión (Model.Conection)

Encargado de conocer los datos del servidor. Dispondrá de funciones de conexión y desconexión así como la ejecución de consultas, las cuales serán totalmente transparentes a él. Recibirá peticiones de consultas y operaciones en lenguaje SQL y devolverá lo que el servidor devuelva. Se trata de un intermediario entre los demás componentes del paquete Modelo y el servidor que aloja la base de datos.

4.2.2.2 Modelo.Identificación (Model.Identification)

Ofrece las funcionalidades de adquisición de información relacionadas con la identificación, es decir, la obtención de datos sobre Corpus, escenarios, muestras, intervenciones y por último frases. Estas funciones las utilizarán tanto el paquete controlador como el paquete vista para adquirir la información a operar o la que ha de mostrar.

4.2.2.3 Modelo.ModeloPatrón (Model.PatternModel)

Análogo al componente anterior. Se encarga de realizar las funciones de adquisición de información relacionada con los datos de los patrones. Esta

funcionalidad consiste en efectuar las sentencias necesarias para permitir al controlador y a la vista la obtención de toda la información referente a los patrones que éstos necesiten como puede ser la obtención de los elementos de un determinado patrón, la obtención de todos los patrones, inserción de patrón, elemento, token, etc.

4.2.2.4 Modelo.ModeloCA (Modelo.CAModel)

Similar a los dos anteriores. Se encarga de ofrecer el acceso a todos los datos referentes a los actos comunicativos que el controlador y la vista necesiten. Las inserciones de este tipo de información las realiza la herramienta COGNOS.CA, pero la obtiene COGNOS.NL para permitir ofrecer la selección de actos comunicativos genéricos. Por tanto este componente ofrecerá funcionalidad de acceso y obtención aunque únicamente esta última sea requerida para este proyecto.

4.2.3 Controlador (Controller)

Este es el paquete encargado de efectuar todas las operaciones necesarias para que el usuario pueda realizar todas las funciones analizadas anteriormente. Se encargará de mantener las ediciones estructuradas hasta que llegue el punto de almacenarlas y gestionará las inferencias cuando el paquete interfaz realice la petición.

4.2.3.1 Controlador.GestorGlobal (Controller.GlobalManager)

El gestor global es el componente que encapsula las ediciones en curso, manteniéndolas en memoria principal hasta que el usuario las guarde. Una edición se guarda en la base de datos individualmente, es decir, una edición de un sub-patrón una vez sea válida, se almacena en la base de datos como si de un patrón completo se tratara. El gestor global se encargará de que el patrón padre tenga una referencia a este sub-patrón. Idénticamente sucederá con la edición de tokens. No se debe de perder la información de los tokens válidos si no se desea completar la edición del patrón al cual se le asigna.

Por consiguiente, este componente realiza una gestión de las ediciones, sus elementos, sus asociaciones con actos comunicativos y otras ediciones a su vez. También gestionará la inserción ordenada de sus elementos.

4.2.3.2 Controlador.GestorFicheros (Controller.FileManager)

Este componente se encarga de ofrecer las funciones necesarias para abrir y leer de un fichero de texto las expresiones. Se encargará por tanto de abrir el fichero solicitado

y extraer siempre la primera intervención, es decir, hasta que se encuentre un salto de línea, y devolverla al paquete de interfaz para mostrarla como expresión editable.

4.2.3.3 Controlador.IntegradorNL (Controller.NLIntegrator)

Este componente tiene como funcionalidad servir de capa entre el paquete NLP y el sistema COGNOS.NL. El paquete suministrado NLP como se comentará más adelante, ofrece la capacidad de inferencia sobre conocimiento, pero este ha de estar en memoria.

El componente NLIntegrator se encarga pues de insertar los patrones que permanecen en memoria en una estructura interna en memoria principal para que el paquete NLP sea capaz de inferir sobre estos y proponer un conjunto de actos comunicativos que encajen con la expresión solicitada. Por consiguiente, este Componente se encargará de:

- Cargar los patrones en memoria. Recopila los patrones editados previamente provenientes de cierto corpus y del conjunto general de la base de datos y los estructura en memoria.
- Limpiar los patrones. Esto será necesario para cuando se deseen cargar nuevos patrones. De esta forma se consigue una independencia cuando se desee inferir dos dominios de interacción distintos.
- Realizar una traducción entre la salida del paquete NLP y los actos comunicativos, es decir, localizar los actos comunicativos propuestos por el paquete NLP y devolverlos a la interfaz para que los muestre.

4.2.4 NLP

Este paquete se encarga de inferir sobre patrones almacenados en memoria principal una expresión dada para así conseguir proponer un conjunto de actos comunicativos. Comprobará por tanto los patrones cargados buscando encajes en sus gramáticas y devolver el conjunto de actos comunicativos asociados a dicho patrón.

Es un componente proporcionado y el propósito en este proyecto es investigarlo, comprenderlo y usar su tecnología para realizar las inferencias sobre el conocimiento almacenado en la base de datos.

En el apartado de implementación se realizará un estudio en detalle de sus estructuras y las equivalencias con los datos del sistema y de qué forma realiza sus operaciones.

4.2.5 JDBC

JDBC son las siglas de *Java Database Conectivity*, y es un API que permite la ejecución de operaciones de bases de datos desde el lenguaje de programación Java. Estas librerías son independientes del sistema operativo donde se ejecute o de la base de datos a la cual se accede. Se utiliza las sentencias en SQL que la base de datos admita.

Para utilizar una base de datos particular, se ejecuta la aplicación junto con la biblioteca de conexión apropiada al modelo de su base de datos (en este caso Oracle), y se accede a ella estableciendo una conexión, para ello es necesario proveer el localizador a la base de datos y los parámetros de conexión específicos. A partir de allí se puede realizar cualquier tipo de tareas con la base de datos a las que tenga permiso: consulta, actualización, creación, modificación y borrado de tablas, ejecución de procedimientos almacenados en la base de datos, etc.

Necesidades de información

Antes de comenzar es necesario aclarar que la base de datos que muestra la Ilustración 4.5 forma parte de un conjunto de bases de datos que en su totalidad completarían la base de conocimiento COGNOS que se muestra en la Ilustración 4.4.

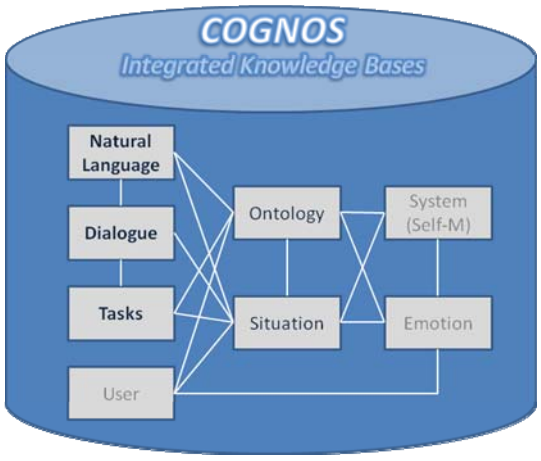


Ilustración 4.4: Base de conocimiento COGNOS. (Calle, Albacete, Sanchez, Del Valle, Rivero, & Cuadra, 2009)

Esta base de conocimiento la integran: la base de datos de lenguaje natural, la base de datos del conocimiento obtenido del diálogo (poblada por COGNOS.Dial). Base de conocimiento de tareas, base de conocimiento del modelo de usuario, ontología de términos, base de conocimiento del modelo de situación, emoción y automodelo. De estas, únicamente las que figuran en negro disponen de soporte actualmente, pero en un futuro se dispondrá de una base de conocimiento totalmente integrada.

La base de datos que alimenta tanto la información visual que se muestra en las interfaces de este proyecto como el motor de inferencia se presenta en la Ilustración 4.5. Dispone de tres apartados significativos totalmente integrados ya que todos son útiles para el proyecto, pero no todos reciben datos de edición por parte de COGNOS.NL. Es en esta división en la que se basará este apartado y seguirá un orden específico según necesidades de información de un apartado sobre otro.

La Ilustración 4.5 muestra el diseño en Entidad-Relación de la base de datos que la aplicación requiere, ya sea para consultar datos como para insertar información en ella. Se analizará su diseño en sucesivos apartados de modo fraccionado debido a la magnitud de ésta. Se dividirá según las funcionalidades que se realizarán sobre esta, a ser: Identificación y selección de Corpus, actos comunicativos y patrones de lenguaje natural.

4.3.1 Apartado de identificación de Corpus.

Este apartado que se muestra en la Ilustración 4.6 es el que entraña las entidades Corpus, Scene, Sample, Intervention y Sentence.

Bajo una visión lógica, todo el conocimiento que se edita para cualquier modelo de conocimiento de este tipo, se engloba bajo dominios de interacción como ya se explicó en el apartado 2.1. Por consiguiente los escenarios pueden ser independientes del corpus y dependientes del dominio de interacción. Esta semántica lógica no está recogida en el diseño y por tanto la entidad Corpus realiza la función de dominio de interacción. En un futuro se mejorará el diseño de la identificación.

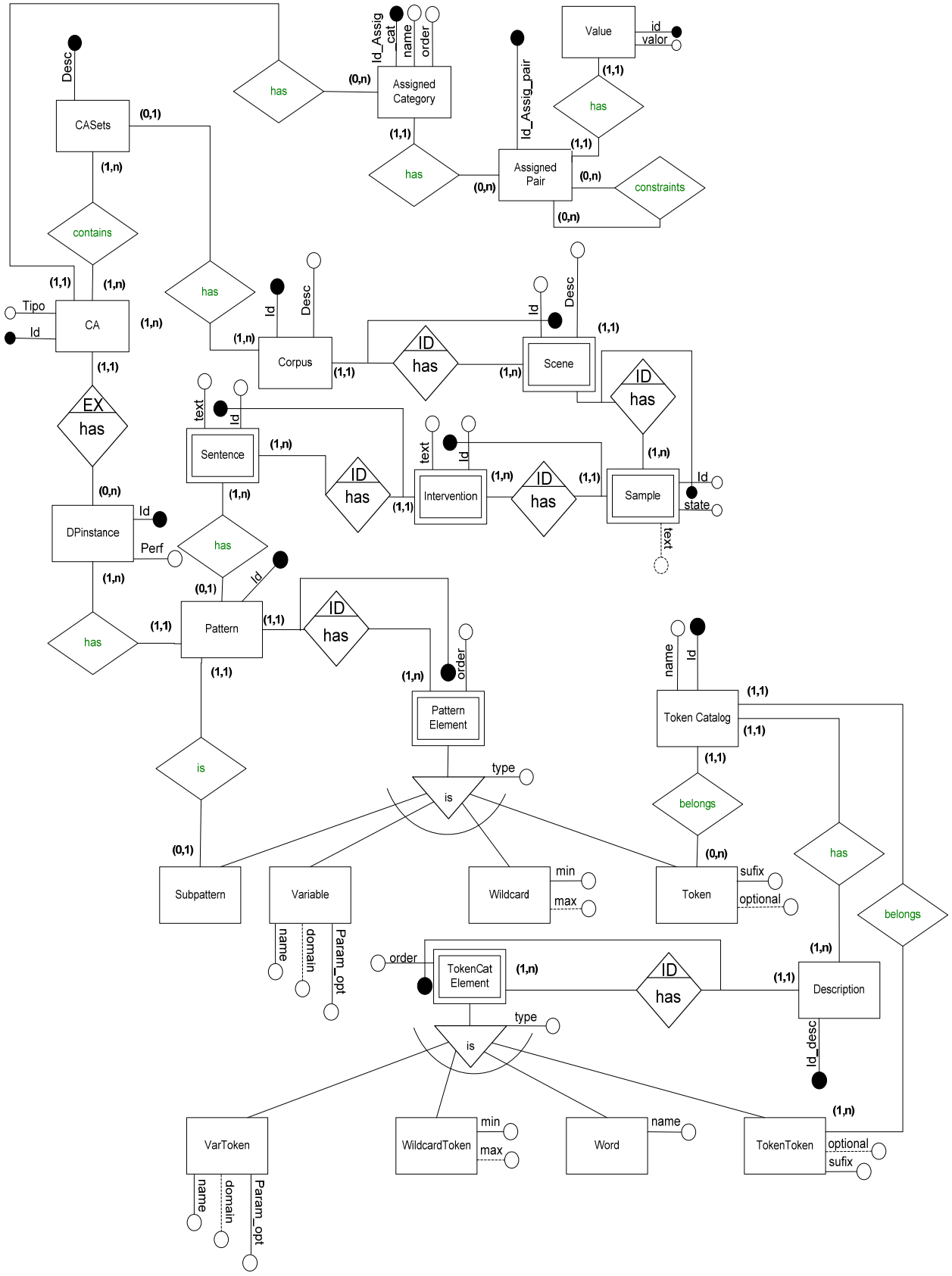


Ilustración 4.5: Base de datos de lenguaje natural.

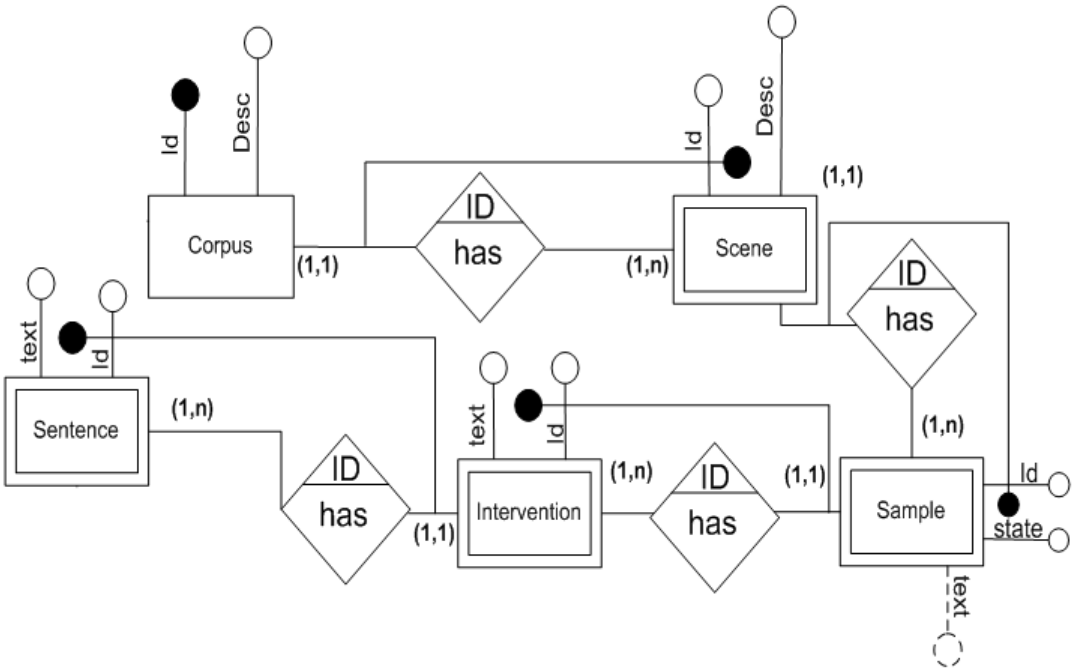


Ilustración 4.6: Detalle del apartado de identificación.

Tanto Corpus como Scene como Sample son entidades de identificación lógica, es decir, se dispone de una muestra de texto la cual ha de pertenecer irremediablemente a un escenario proveniente de un corpus para que dicha muestra aporte conocimiento. Si una muestra de diálogo no está identificada en un dominio de interacción, no constituye por sí sola información o conocimiento extraíble útil.

Por ello la lógica de este fragmento es que un Corpus puede contener varios escenarios y que un escenario puede contener varias muestras. Es en la entidad muestra en la que se almacena el fragmento de diálogo al cual pertenecerán las intervenciones y cada intervención contendrá frases.

Se trata de cuatro relaciones débiles en identificación por lo comentado anteriormente, una intervención carece de información si no está asociada e identificada con un corpus concreto pero una expresión no tiene porqué. De forma similar ocurre lo mismo con el resto de entidades en este apartado de la base de datos con la excepción de la restricción lógica comentada en párrafos anteriores.

Este apartado es accedido por el sistema de este proyecto para efectuar a identificación de la expresión que se desea editar. La información referente a Corpus, Scene, Sample, Intervention y Sentence será poblada por la aplicación COGNOS.Dial

según ésta vaya anotando cada uno de estos diálogos, y de esta manera poder extraer las expresiones de los diálogos anotados por Dial

Se deja como línea futura ampliar la base de datos para que recoja la posibilidad de asignar análisis generales que no pertenezcan a ningún corpus en concreto. Actualmente para paliar esta carencia, se creará un corpus general perteneciente a la colección de corpus del sistema.

4.3.2 Apartado de actos comunicativos genéricos.

Este sub-apartado de la base de datos mostrado en la Ilustración 4.7 es el encargado de abstraer los actos comunicativos y representar una colección de posibilidades de actos comunicativos editada mediante la herramienta COGNOS.CA. Es parte del editor de lenguaje natural mostrar los datos pertenecientes a esta sección de la base de datos para que el usuario, a partir de una definición de acto comunicativo genérico, seleccione un acto comunicativo concreto y asociarlo a la expresión que se está editando.

Está estrechamente relacionado con Corpus, ya que los actos comunicativos se estructuran mediante conjuntos de los mismos y cada uno de estos conjuntos está asociado uno o varios Corpus. De esta forma un corpus especificado solo dispondrá de un conjunto de actos comunicativos.

Un acto comunicativo está formado, como ya se ha comentado anteriormente, por un tipo y una secuencia de categoría valor. En este apartado de la base de datos se modelan para un tipo de acto comunicativo, sus posibles categorías así como sus posibles pares categoría valor. Se gestiona mediante la interrelación Constraints (restricciones) el hecho de que según qué pares categoría-valor existan anteriormente, serán válidos unos pares u otros.

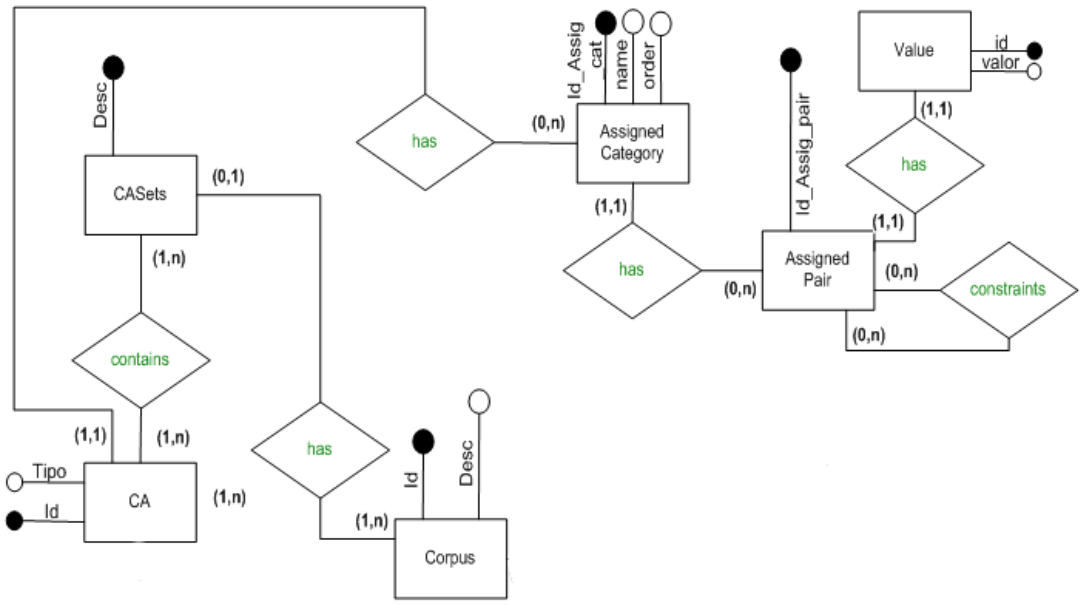


Ilustración 4.7: Detalle del apartado de actos comunicativos.

Las categorías se identifican tanto por un identificador interno y transparente al usuario como por su nombre. De este modo, dos categorías con el mismo nombre podrán contener valores distintos y estar asociadas a distintos actos comunicativos.

4.3.3 Apartado de patrones de lenguaje natural.

Este es el apartado que edita la herramienta de edición de lenguaje natural y que a su vez es consultado para así poder realizar las inferencias sobre patrones editados previamente. El detalle en entidad-relación es el que se muestra en la Ilustración 4.8.

En primer lugar y como conexión con el apartado de actos comunicativos existe la entidad llamada DPInstance, es decir, Instancia de pieza de discurso. Contiene la referencia a un acto comunicativo concreto pero asociado a un acto comunicativo abstracto perteneciente a la base de datos. Contiene como atributo una cadena de texto en el cual aparecerá la forma del acto comunicativo seleccionado y asignado al patrón. Esta cadena siempre seguirá el siguiente formato:

<tipo de CA>(<nombre de categoría1> = <Valor seleccionado1>, <nombre de categoría2> = <Valor seleccionado2>, …, <nombre de categoría N> = <Valor seleccionado N>)

Se ha diseñado un formato para las instancias de actos comunicativos en lugar de haberlo desglosado en entidades debido a que estos actos comunicativos concretos siempre se almacenan y se recuperan a la vez, si se hubiese diseñado en entidades

hubiera sido necesaria la implementación de mecanismos de control extras. Estas instancias de actos comunicativos funcionan como una caja negra de cara al sistema una vez editados, por tanto se simplifica habiéndolas diseñadas de esta forma.

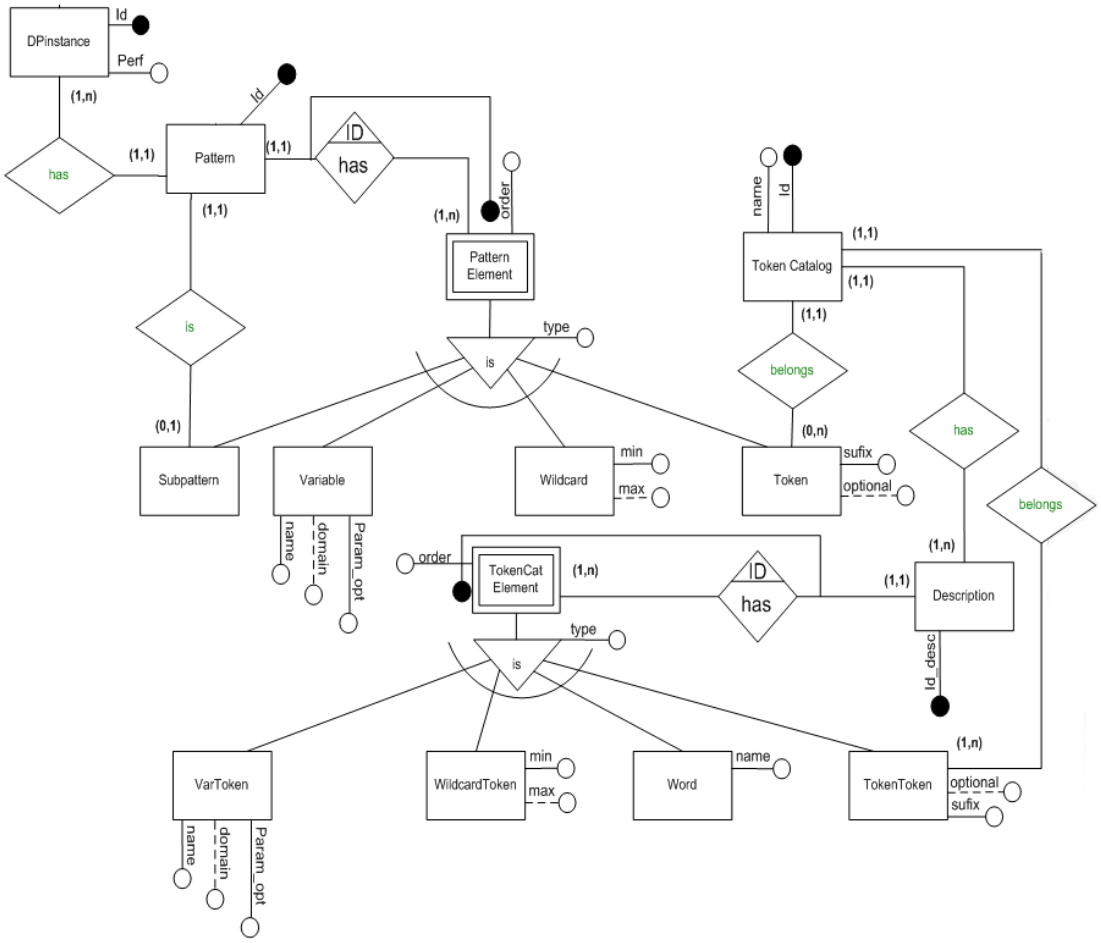


Ilustración 4.8: Detalle del apartado de la información de patrones.

Como se puede apreciar, esta entidad DPInstance está asociada a la entidad Pattern (Patrón), el cual a su puede representar el patrón de varias frases del Corpus. La entidad Pattern es la clave de este módulo de la base de datos completa ya que es tanto creado por la herramienta de edición de lenguaje natural como obtenido por parte del componente de integración con el motor de inferencias.

Un patrón puede ser editado con independencia de una frase identificada en corpus editados, y es necesario almacenar dicho análisis. El modelo relacional no recoge dicha posibilidad, ya que es necesario que un patrón esté asociado con una frase de un corpus específico. En la implementación física se realizará dicha posibilidad. Quedará pues para un futuro realizar un diseño que permita la asignación de ediciones a corpus o a frases independientes.

Un patrón contiene cierto número de elementos, los cuales son representados por la entidad débil PatternElement. Cada uno de estos elementos puede ser, como se explicó anteriormente, un patrón (sub-patrón), una variable, un comodín (wildcard), o un token.

Se necesita un repositorio de tokens los cuales se almacenan en la entidad tokenCatalog (catalogo de tokens) a la cual pertenecen todos los tokens creados pudiendo repetir su nombre. Cada uno de estos tokens presentes en esta colección dispondrá de un cierto número de descripciones las cuales estarán formadas a su vez por elementos. Los cuales podrán ser: variables (provenientes de token), comodines (wildcardToken), palabras (Word) o a su vez otro token proveniente del catálogo de tokens.

De esta forma se representa que un token dispone de distintas opciones de encaje llamadas descripciones y representadas por sus elementos. Lo que realmente se asocia a un patrón son los tokens del catálogo, sin embargo, el componente que obtiene esta información para inferir sobre ella, obtiene todas las descripciones que contiene el token asignado a la expresión.

Un token y todos sus elementos editados son independientes del patrón en el cual se editaron inicialmente, por tanto es posible que otras ediciones lo contengan. Si este token contiene elementos variables Estas variables han de ser asumidas por el primer patrón (ó sub-patrón según corresponda) para que, de esta forma, puedan ser asignadas a actos comunicativos. Existe un problema de espacio de nombres en este caso, ya que se pierde la independencia cuando un patrón 'hereda' una variable y éste la tiene que gestionar. Para solventar este problema se ha incluido un atributo sufix en el elemento token de patrón (por tanto, en sub-patrón también). El nombre de las variables provenientes de este token que el padre deberá de gestionar, contendrán este sufijo a continuación de su nombre editado. De esta forma se mantiene la independencia de todos los tokens con respecto a los patrones que los usen.

Para eliminar redundancias se ha eliminado una asociación lógica entre variable y categoría, ya que en el preformativo aparecería el nombre de la variable en lugar del valor siempre y cuando la categoría pueda tener valores variables.

Diseño de la interacción

En este apartado se diseñarán los mecanismos para la interacción con el usuario. Los principales usuarios a los que va dirigida la herramienta serán expertos en lingüística y, posiblemente, no tengan gran experiencia con aplicaciones informáticas. Por ello se ha de diseñar las interfaces de tal modo que faciliten su uso por parte de los usuarios: deberán de ser intuitivas, amigables y sencillas.

A continuación se mostrarán las maquetas de las interfaces de usuario las cuales son un acercamiento a la implementación. Estas son prototipos sin funcionalidad, para de esta forma, facilitar la implementación. Se realizarán las ilustraciones y la secuencia entre distintas interfaces comentando qué pasos ha de realizar el usuario para pasar de una a otra y las funciones de información hacia el usuario que se acometen.

Se especificarán todos los diseños para la inserción de los datos más relevantes así como los mecanismos de respuesta hacia el usuario por parte de la interfaz. A su vez se justificará todas y cada una de las decisiones de diseño en base a los requisitos, ya sean funcionales o referentes a la amigabilidad de la herramienta.

La aplicación consta principalmente de dos interfaces principales, edición de patrón y edición de token. Se podría incluir a estas dos la interfaz de edición de sub-patrón para una mejora en la comprensión del flujo de interacción, aunque ésta sea prácticamente idéntica a la edición de patrón.

Los siguientes apartados comentarán cada una de las tres interfaces presentes en el sistema. Cada uno de ellos explicará todos los detalles presentes en las interfaces, así como sus posibilidades y, en su caso, carencias que se mejorarán en un futuro.

4.4.1 Interfaz de edición de patrón raíz.

Esta es la pantalla principal de la herramienta en la cual se editará la gramática relajada de una expresión de lenguaje natural dada. En primer lugar y con diferencia de la interfaz de edición de sub-patrones, presenta los selectores de Corpus, Escenario, Muestra, Intervención y Frase (Ilustración 4.9). Todos los selectores posteriores a uno dado, únicamente mostrarán sus elementos en relación con los seleccionados anteriormente.

Ilustración 4.9: interfaz de identificación

En la Ilustración 4.10 se puede apreciar el área en el que inicialmente aparece la expresión de lenguaje natural que se va a ir editando. A medida que el usuario vaya anotando elementos, éstos aparecerán entre los caracteres "<" y ">" y mostrando la etiqueta que el usuario utilizó para nombrar ese elemento. Por otra parte se exhibe subrayado cada tipo de elemento de un color, para de esta forma conocer el tipo de elemento que se corresponde de manera ágil.

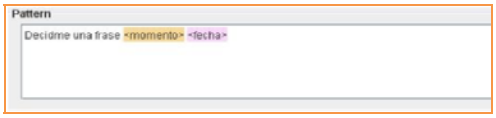


Ilustración 4.10: campo de expresiones.

Como se ha comentado anteriormente, un elemento importante de las gramáticas son las variables, las cuales se mostrarán junto con su dominio y asignaciones a actos comunicativos en la tabla que aparece en la Ilustración 4.11. Se puede anotar como variable cualquier cadena de caracteres, pero éstas perderán su valor y tomarán el valor del dominio asignado. Actualmente solo se da soporte a tres dominios que son: fecha, hora y cantidad. Estos tres valores se podrán asignar a la variable mediante el botón colocado expresamente para ello. Mediante otros tres botones se permitirá realizar la eliminación de variables, la asignación de una variable a un acto comunicativo (antes solicitará la selección de la categoría a la cual se le asignará) y la anulación de la asignación a los actos comunicativos que la variable pueda estar asignada.

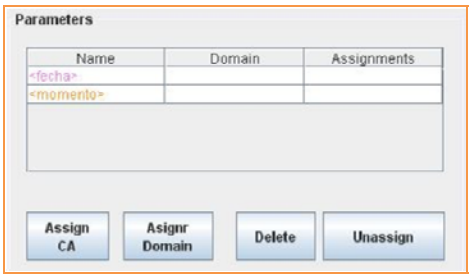


Ilustración 4.11: interfaz de gestión de

Un patrón (o sub-patrón) puede tener asignados varias instancias de actos comunicativos, para ello se facilita un selector de los mismos (Ilustración 4.12) en el

cual, aparecerán en un principio todos los tipos de actos comunicativos pertenecientes al conjunto del corpus del cual proviene la expresión en edición. En caso de que no pertenezca a ningún corpus, se cargará la totalidad de actos comunicativos. Una vez el usuario seleccione el tipo de acto comunicativo aparecerán tantas listas como categorías asignadas tenga, y en cada lista todos los valores. A medida que el usuario vaya seleccionando valores de las categorías, en las categorías sucesivas únicamente se mostrarán aquellos valores que se hayan editado como válidos (Mediante COGNOS.CA). Cada uno de los actos comunicativos seleccionados aparecerá en la lista de la Ilustración 4.13 con el formato definido para ellos que corresponde con el valor almacenado en la base de datos.

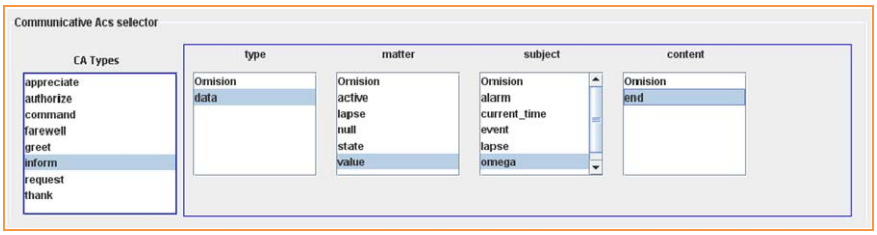


Ilustración 4.12: selector de actos comunicativos.

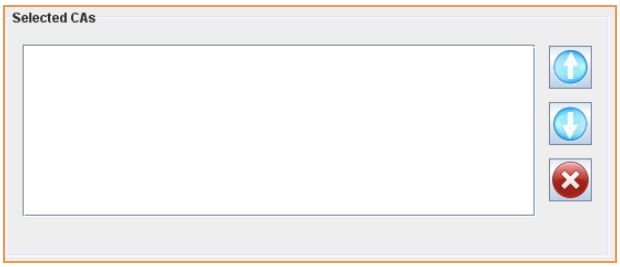


Ilustración 4.13: lista de actos comunicativos.

El control de inferencias se realiza mediante un botón (Ilustración 4.14) el cual activará tanto la carga de patrones como la primera inferencia de la expresión. Existen deshabilitados un par de botones “derecha” e “izquierda” los cuales fueron propuestos como soporte a líneas futuras.

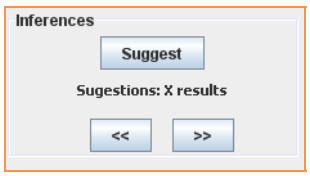


Ilustración 4.14: Control de inferencia.

En la Ilustración 4.15, se muestra el panel informativo de la pantalla principal, el cual informará al usuario de si las operaciones se han realizado satisfactoriamente, se ha

producido un error (en cuyo caso aparecerá texto en color rojo) ó simplemente mostrando información de, por ejemplo, la ruta del fichero que se ha cargado.

Ilustración 4.15: panel informativo.

La interfaz contiene los controles de anotación, que figuran en la Ilustración 4.16. Estos sirven para seleccionar el tipo de elemento que corresponde a una selección dada. Para este apartado estos elementos serán: Sub-patrón, token o token opcional, variable o variable opcional y comodín.

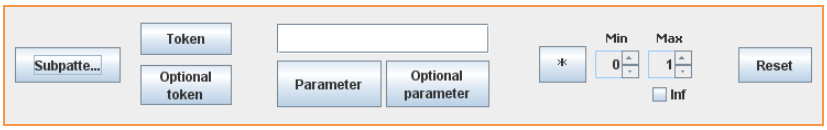


Ilustración 4.16: Edición de elemento.

- Sub-patrón. Si un elemento se seleccionado y anotado como sub-patrón, se abrirá una nueva ventana de edición de sub-patrón. El resultado para la interfaz una vez se haya editado por completo el sub-patrón, es la aparición entre los caracteres "<" y ">" y subrayado en color azul el nombre que se haya asignado al sub-patrón.

- Token o token opcional. Se abrirá una ventana para realizar la edición o reutilización según casos del token correspondiente. La etiqueta del token seleccionado o editado aparecerá entre los caracteres "<" y ">" y subrayado de color verde si se trata de token o en color morado si se trata de token opcional.

- Variable o variable opcional. Será necesario para editar una selección como variable, que el usuario especifique una etiqueta, el cual será el nombre de la variable. Una vez esto la variable aparecerá entre los caracteres "<" y ">", subrayada en un color rosa si se trata de variable normal y naranja si es optativa, y en el mismo color en el cual aparezca subrayada, en la tabla de variables que figura en el la Ilustración 4.11.

- Comodín. Para editar una selección como comodín es necesario especificar el número mínimo y máximo de palabras de las que consta. Si el número mínimo es 0, significará que en esa posición no debe de aparecer ninguna palabra y si el número máximo es *inf* significará que podrá aparecer un número indeterminado de ellas.

La Ilustración 4.17 recoge la maqueta de la interfaz general de edición del patrón, incluyendo todos los elementos que se han expuesto anteriormente de modo desglosado.

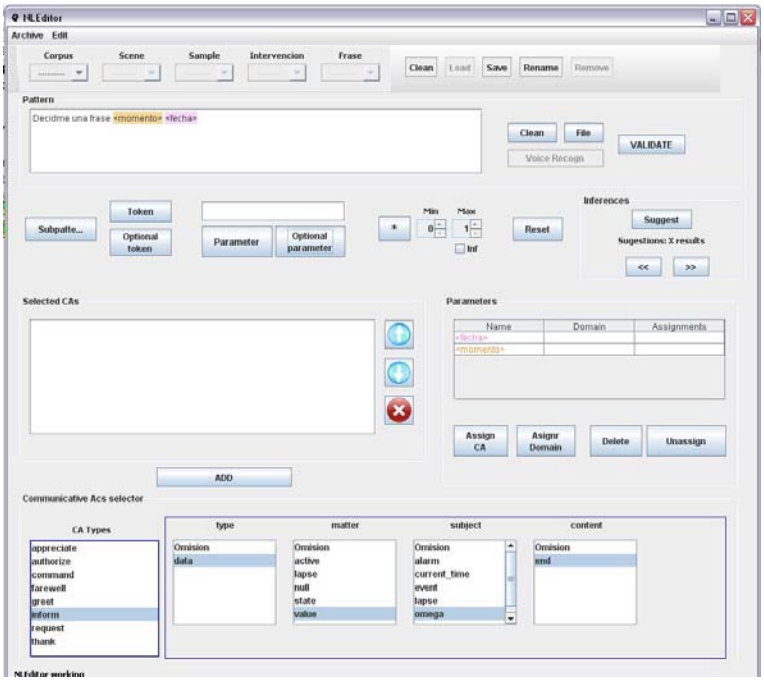


Ilustración 4.17: Interfaz de edición de patrones.

4.4.2 Interfaz de edición de sub-patrones.

La interfaz de edición de sub-token contiene los mismos elementos que la interfaz principal salvo los controles de identificación, que carece de ellos. El motivo de esta omisión es que esta interfaz recibe la expresión a anotar por parte de la edición principal (fragmento de texto anotado como sub-patrón). En lugar de esta barra de identificación, dispone de una zona para asignar una etiqueta al sub-patrón editado y para aceptar o cancelar la edición. En la Ilustración 4.18 se puede apreciar esta zona junto con una visión general de la interfaz

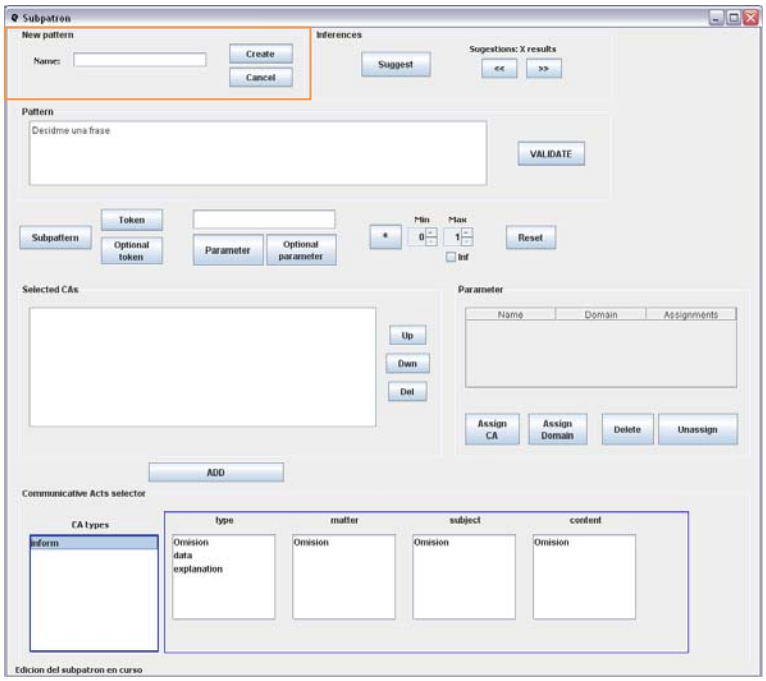


Ilustración 4.18: Interfaz de edición de sub-patrón.

4.4.3 Interfaz de edición de tokens.

La interfaz de edición de tokens contiene un mismo cuadro de texto en el cual se muestra la expresión a formalizar como token, similar al de la edición de sub-patrón en el que los elementos anotados aparecen entre los caracteres de marcado y subrayados conforme al mismo criterio citado en el apartado anterior. Este texto ya no se puede editar, únicamente se permitirá al usuario seleccionar fragmentos.

En la Ilustración 4.19 se aprecian los controles de anotación idénticos a los de la interfaz de edición de patrón con la salvedad de que no se encuentra presente el botón sub-patrón ya que los tokens no pueden tener estos elementos. Referir al apartado 4.4.1 para mayor información acerca de los controles.

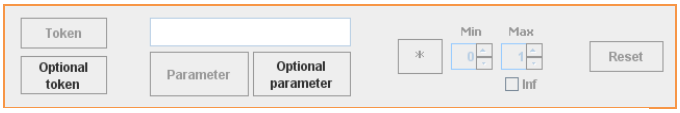


Ilustración 4.19: controles de edición de token.

La interfaz de edición de token permite crear o reutilizar tokens, para permitir la creación existe un campo en el cual se deberá de introducir el nombre del token nuevo (Ilustración 4.20) y a continuación se editará el fragmento de expresión añadiendo elementos si procediese. Como ya se ha comentado, se tomará como elemento “palabra” (word) aquellas cadenas que no hayan sido anotadas.

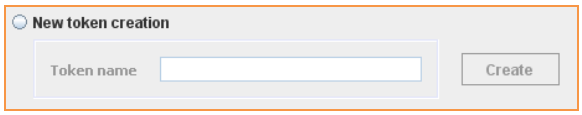


Ilustración 4.20: nuevo token.

Si por el contrario se desea reutilizar un token se podrá escoger cuál en la lista que aparece en la Ilustración 4.22 una vez seleccionado un token, aparecerán sus descripciones en la lista que se aprecia en la Ilustración 4.21. si se marca la casilla inferior a esta lista, se reutilizará por completo el token, y si no se ha hecho así, se deberá de anotar los elementos de la expresión para formar una nueva descripción, la cual se añadirá al token seleccionado.

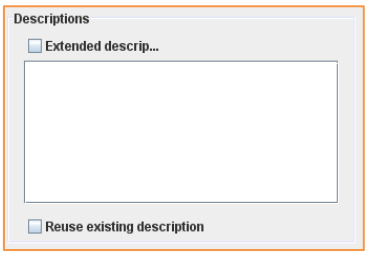


Ilustración 4.21: lista de descripciones.

Ilustración 4.22: Lista de tokens existentes.

4.4.4 Flujo de pantallas.

La interfaz principal es la edición de patrones y es con la que se inicia el sistema. Esta interfaz puede abrir tanto ediciones de sub-patrón como ediciones de token, como puede apreciarse en el diagrama de flujo de pantallas de la Ilustración 4.23. Siempre y cuando una ventana nueva invoque a una edición de un nivel inferior, la ventana padre se congelará esperando el retorno de la edición en curso (de nivel inferior).

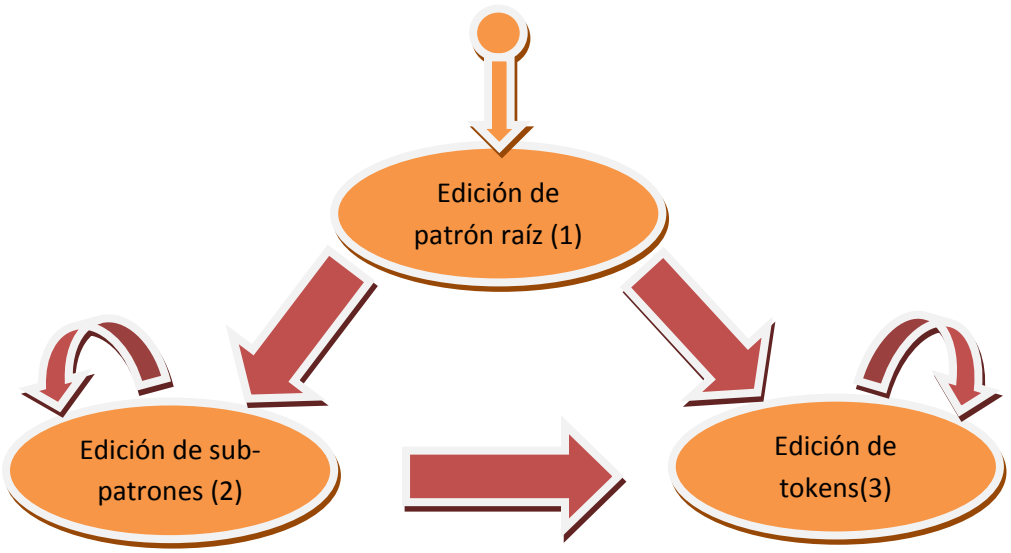


Ilustración 4.23: Flujo de pantallas.

La interfaz de edición de sub-patrones al poder contener elementos que son a su vez otros sub-patrones, puede abrir otras ventanas de edición de sub-patrón así como de edición de token. Cuando se ha realizado una edición válida de un sub-patrón, este devuelve sus datos al elemento sub-patrón o patrón raíz que le invocó.

Por último, un elemento token puede estar presente tanto en el patrón raíz como en los sub-patrones e incluso en otros tokens por tanto las interfaces de edición de patrón raíz, sub-patrón y token pueden instanciar una nueva edición de token.

5 Implementación

En este apartado se describirá sistema desarrollado a nivel de implementación. Seguirán las pautas del apartado anterior, profundizando en el nivel de detalle de cada uno de sus componentes. En primer lugar se especificará la implementación física del sistema, cómo utiliza los recursos especificados en los requisitos y los beneficios de las decisiones tomadas. En segundo lugar se detallará en profundidad el sistema cliente, se ilustrarán las explicaciones con los diagramas de clases del sistema. Por último se profundizará en el diseño de información a bajo nivel realizado y las decisiones de transformación tomadas para así poder contemplar toda la semántica expuesta en el apartado anterior.

La aplicación se ha desarrollado en el entorno de programación para Java NET BEANS 6, el cual proporciona una mejor gestión del diseño de interfaces, que es, junto con la información almacenada, el punto que mayor esfuerzo de implementación requiere. El utilizar un entorno de desarrollo para la implantación de aplicaciones software agiliza la ejecución de pruebas, facilita la depuración y automatiza la compilación, proporciona a su vez utilidades gráficas para la inclusión de librerías externas y control de versiones.

Se ha instalado un servidor del controlador de versiones Collabnet Subversion cuyo uso principal es el de realizar copias de seguridad semi-automáticas. También posibilita retroceder a versiones anteriores del sistema, si bien en el desarrollo de este proyecto esta funcionalidad no ha sido utilizada.

Arquitectura física del sistema

Se especificó previamente que el sistema implementa una arquitectura cliente - servidor. En este caso, el servidor contiene la base de datos central que es poblada de conocimiento por las diferentes instancias de la aplicación. Este servidor está alojado en

una máquina que posee un procesador de doble núcleo AMD Opteron (™) a 2.21 GHz y 4 GB de memoria RAM. Características más que suficientes para permitir un funcionamiento óptimo del sistema.

Es posible que existan dificultades en cuanto a la conexión del servidor dependiendo de la red en la que se encuentren. El sistema gestor de bases de datos está configurado para escuchar por el puerto 1521 las conexiones a las distintas bases de datos que posea. Por tanto, es necesario configurar cualquier elemento de bloqueo de conexión como pueden ser los firewall para que permitan el acceso a dicho puerto del servidor.

Descripción detallada de clases.

En este apartado se describirán las clases que consta la aplicación cliente así como los detalles de implementación y las funciones más relevantes realizadas. Se acompañarán las explicaciones con diagramas UML que ilustrarán las relaciones entre los distintos elementos implementados. Se ha utilizado el programa Altova UModel mara realizar los esquemas presentes en este apartado.

Es importante señalar que en los diagramas de clases que se mostrarán en los siguientes sub-apartados únicamente aparecerán los atributos y los métodos más relevantes, de tal manera que el lector llegue a comprender las principales decisiones de implementación que afectan a la funcionalidad del sistema. Por lo tanto, se excluirán de los diagramas de clases aquellos atributos y métodos que no se considera que realicen ninguna aportación a la comprensión de la implementación del sistema.

5.2.1 Paquete Vista.

Este paquete contiene las implementaciones de las interfaces de usuario detalladas en el capítulo 4.4. El usuario edita las expresiones en la interfaz principal, que corresponde a la edición del patrón raíz, una vez continúe la edición, recorrerá las diferentes interfaces de edición como son las de sub-patrón

En la Ilustración 5.1 se muestran las diferentes clases que elaboran el paquete vista. Por un lado está la clase principal, que corresponde con la primera toma de contacto con el usuario. Se trata pues de la interfaz de edición de patrón raíz. Esta clase contiene una colección de CategoryPanel, los cuales son los encargados de mostrar la información de

cada categoría en el selector de actos comunicativos. La clase FrameSubPattern se encarga de implementar la interfaz de edición de sub-patrones y la interfaz de edición de tokens es gestionada por la clase FrameTokenizer. Todas estas serán detalladas en los siguientes apartados.

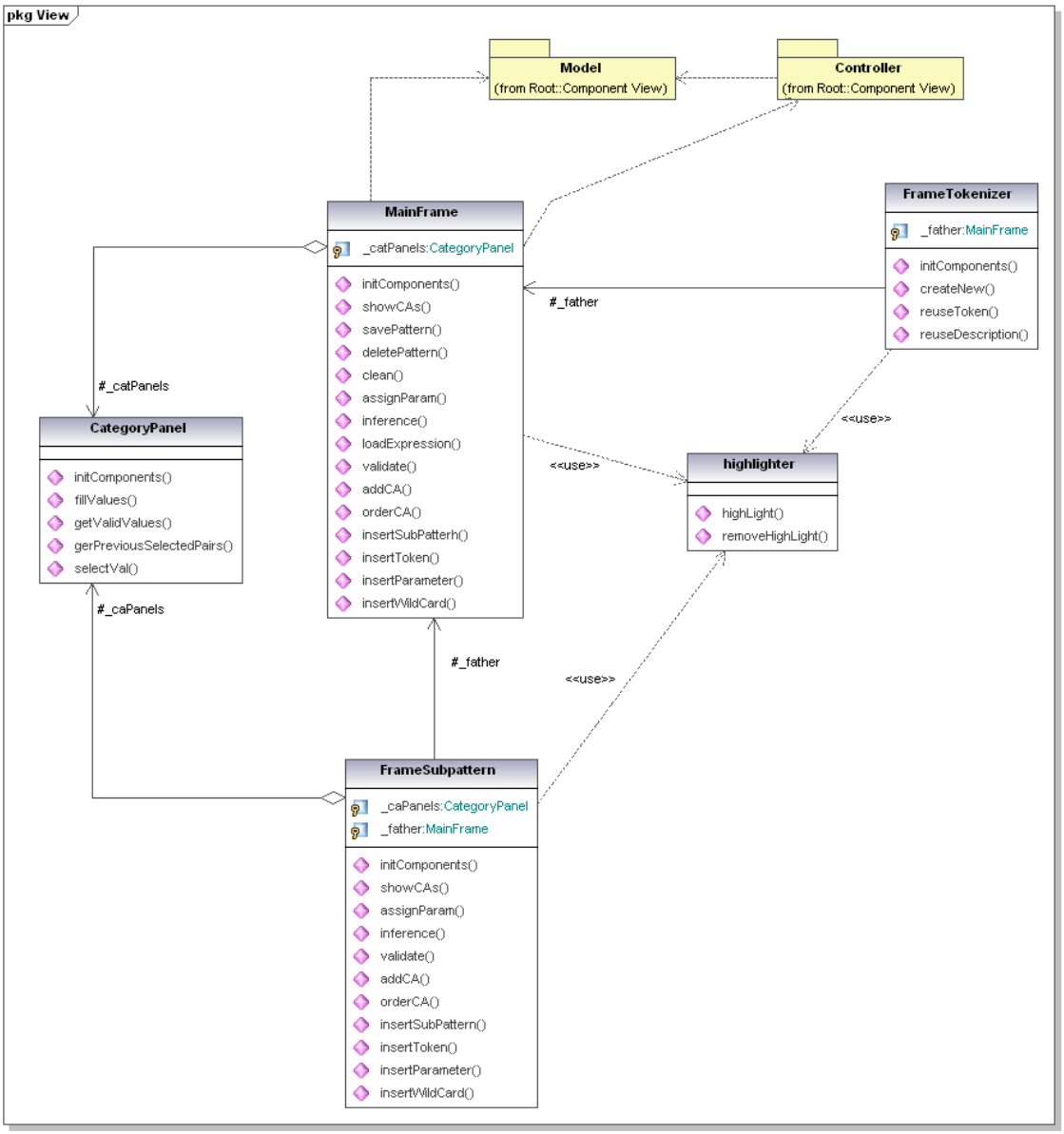


Ilustración 5.1: Diagrama de clases del paquete Vista.

Para realizar la implementación de los componentes de los que constan las interfaces se ha utilizado del editor de interfaces que proporciona el entorno de desarrollo NetBeans. Utilizar un editor de interfaces reduce considerablemente el tiempo y la complejidad de la creación de estos componentes. Este es el paquete que más decisiones de diseño y comportamiento visual contiene. Por tanto solo se comentarán las funciones implementadas más características funcionalmente hablando.

Se han obviado los atributos relacionados con los elementos gráficos que estas clases presentan, para ganar en simplicidad y comprensión.

5.2.1.1 MainFrame.

Esta clase implementa la interfaz de edición del patrón raíz, por tanto incluye los eventos capturados para cada elemento de la interfaz y se dotarán de las llamadas al paquete controlador, para que efectúe de este modo el comportamiento deseado. Estos eventos realizarán, por tanto, las funciones que figuran en la Ilustración 5.2.

El método *initComponents()* presente en todas las clases del paquete de interfaces es creado por el entorno de desarrollo y es el encargado de inicializar todos los componentes visuales de los que conste la clase, así como colocarlos y empaquetarlos en el marco visual según estén diseñados.

El método *showCAs()* es el encargado de obtener y mostrar los actos comunicativos que corresponden. Esta acción se lleva a cabo al inicio de la aplicación por primera vez y siempre que se cambie la selección de corpus. Los actos comunicativos que este método muestra, depende, por tanto, de los corpus seleccionados; en caso de no haber seleccionado ningún corpus, los actos comunicativos son la totalidad de los presentes en la base de datos. Si por el contrario existe un corpus seleccionado, los actos comunicativos serán los presentes en el conjunto editado para ese corpus. Una vez seleccionado un acto comunicativo, se mostrarán sus categorías en paneles independientes de la clase CategoryPanel que representan las categorías editadas para ese acto comunicativo en concreto. Cuando el acto comunicativo esté completamente seleccionado, se añade el acto comunicativo a la selección de la edición mediante el método *addCA()*.

El método encargado de realizar las funciones de proposición de actos comunicativos es inference(), que se encarga de que el componente del paquete controlador correspondiente realice cotanto la función de carga de conocimiento en memoria principal como la inferencia en sí.

Existen métodos de almacenamiento y modificación. Como *son savePattern()* y *deletePattern()*. Este último realiza la limpieza del patrón en edición. No se ha implementado la posibilidad de eliminar un patrón previamente editado con éxito, esta posibilidad se contemplará en implementaciones y mejoras futuras.

Los métodos *insertSubPattern()*, *insertToken()*, *insertParameter()* e *insertWildCard()*se encargarán de insertar visualmente los elementos seleccionados como elemento del tipo que corresponda, resaltándolo, mediante la clase HighLighter que se explicará más adelante y gestionando las selecciones realizadas con anterioridad. El método *clean()* eliminará todas las selecciones y ediciones realizadas para dicho patrón en un momento dado.

Ilustración 5.2: Edición principal.

Existen funciones no indicadas, que son los *pop-ups* informativos. Estas funciones se resuelven mediante invocaciones a otro objeto proporcionado por las librerías gráficas de Java, que se ha adaptado para que cumpla con las necesidades específicas. La siguiente ilustración muestra un ejemplo de creación de *pop-ups* sencillos, se trata de la selección de dominio para las variables editadas:

```
Object[] domains = {"HORA", "FECHA", "CANTIDAD"};
String s = (String)JOptionPane.showInputDialog(this,
                "Select a domain:\n",
                "Domain assignment",
                JOptionPane.PLAIN_MESSAGE,
                null,
                domains,
                "HORA");
```

Ilustración 5.3: código de un optionPane.

5.2.1.2 FrameSubPattern.

Esta clase está implementada de forma similar a la anterior (MainFrame) y constituye un subconjunto de funciones de éste. Por este motivo se ha reutilizado gran parte del desarrollo anterior. Contiene una diferencia con respecto a la interfaz principal consistente en la identificación: en lugar de permitir la identificación o la escritura manual de una expresión, ha de obtener la expresión editada en la interfaz MainFrame como sub-patrón. Por tanto en el diagrama del paquete View, ésta clase contiene el objeto de la clase que le haya invocado, para así poder tener referencia a éste en caso de necesidad, por ejemplo, cuando se ha acabado la edición del sub-patrón completo, el editor superior ha de tener su referencia.

5.2.1.3 CategoryPanel.

Este panel es el encargado de mostrar la información de cada categoría de un acto comunicativo concreto. La funcionalidad de este componente es dependiente de los objetos que le contengan, ya que para mostrar los valores de una cierta categoría, es necesario conocer los valores anteriores seleccionados anteriormente, de esto se encarga el método *getPreviousSelectedPairs()*, que obtiene los datos sobre los paneles anteriores que posee el objeto que está usando cada instancia concreta. Los valores que muestra esta clase en una JList son los válidos para los pares seleccionados previos.

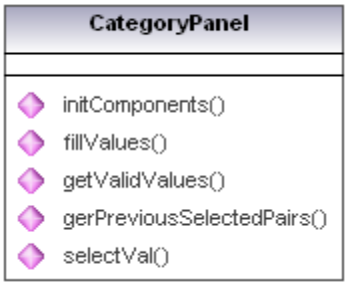


Ilustración 5.4: Clase de edición de categoría

En el momento en el que la clase MainFrame añada el acto comunicativo, se obtendrán los valores seleccionados para cada panel de categoría y se formará el String que representa el acto comunicativo concreto.

5.2.1.4 FrameTokenizer.

Esta clase implementa la interfaz de edición de tokens. Es invocada cuando se edita como token un elemento en las dos clases anteriores o en una edición de token. Al igual

que FrameSubPattern, bloquea cualquier edición hasta que el token se haya editado con éxito.

Como se describió en el análisis, un token puede contener otros tokens (opcionales o no), variables (opcionales o no), y comodines. El usuario decide si quiere reutilizar un token, reutilizar una descripción o anotar un token nuevo, cada una de estas funcionalidades las realiza los métodos *reuseToken()*, *reuseDescription()*, *createNew()* respectivamente.

Un token es totalmente independiente del dominio de interacción por tanto se puede reutilizar tokens provenientes de otros dominios. Por consiguiente, esta clase muestra mediante un objeto JList la totalidad de tokens (tokens del catálogo) presentes en la base de datos para que el usuario escoja uno de ellos. Como esta colección de tokens puede llegar a ser de unos cuantos cientos de elementos, se ha pensado realizar en un futuro búsquedas en tiempo real mediante un campo de introducción en lugar de una lista.

Esta clase consta a su vez de otras dos listas que muestran, según el token seleccionado sus variables y todas las descripciones que lo forman. Cuando el usuario desee reutilizar completamente un token (sin tener que anotar la expresión) podrá también reutilizar descripciones. En caso de que quiera insertar una nueva descripción, seleccionará de la lista de tokens a la que desearía añadir la descripción y anotar la expresión.

5.2.1.5 HighLighter.

Esta clase implementa las utilidades que utilizan las tres interfaces de edición para subrayar texto. Cada instancia de esta clase controlará las posiciones de texto subrayadas y aporta los métodos necesarios para realizar el resaltado de un elemento de su color respectivo.

Para realizar un resaltado, esta clase recibe las posiciones del texto a resaltar y las almacena junto con el color (que depende del tipo de elemento). De esta forma cada vez que un nuevo elemento y debido a las restricciones de los objetos de intrerfaz de las librerías gráficas de Java, es necesario volver a resaltar los fragmentos del texto ya editados de nuevo.

5.2.2 Paquete Controlador.

Este paquete implementa las estructuras en memoria y las operaciones necesarias para disponer de una edición. Se entiende edición como la edición de un único token, patrón o sub-patrón. Por tanto este paquete gestiona la edición en curso. Cuando una edición es válida se insertará el conocimiento en la base de datos. En la Ilustración 5.5 se aprecian las relaciones internas y externas. La clase que la vista utiliza para gestionar las ediciones es EditionElement y la parte de la conexión con el paquete modelo se realiza a través de la clase CtrlPatternElement. En los sucesivos apartados se detallarán las diferentes clases y su funcionalidad principal.

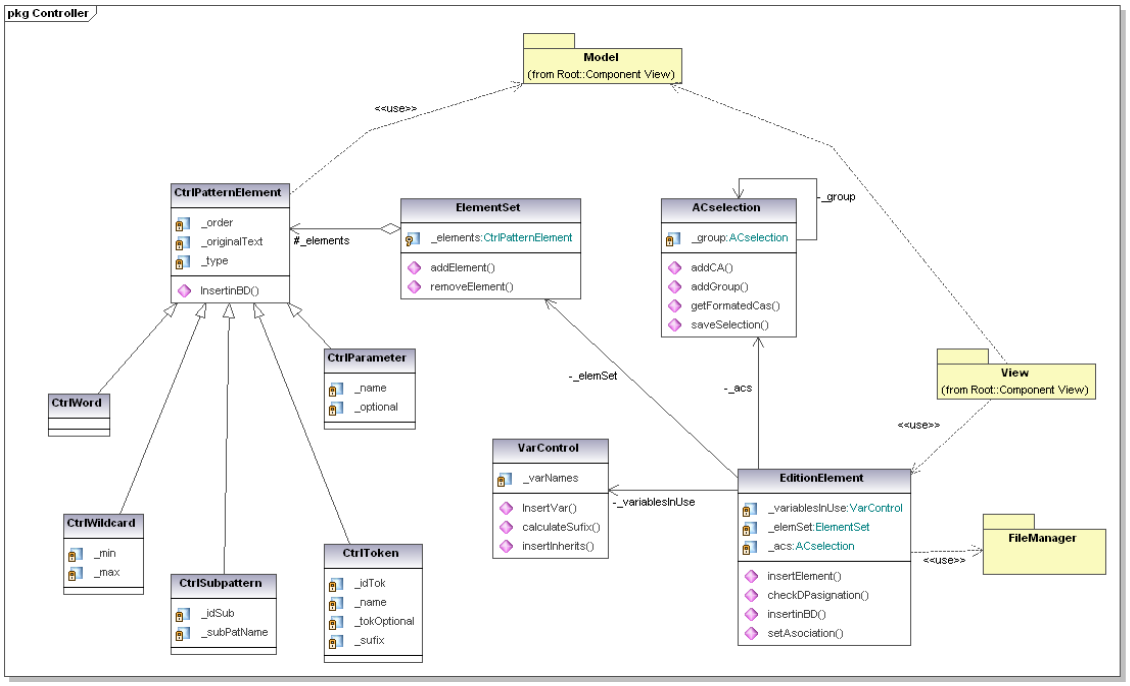


Ilustración 5.5: Diagrama de clases del paquete Controlador.

5.2.2.1 EditionElement.

Esta clase (Ilustración 5.6) representa una edición de cualquier elemento (patrón, sub-patrón o token) como se ha comentado en el punto anterior. Contiene pues instancias de las clases VarControl, ElementSet y ACselection. Un token no contiene selecciones de tokens, por tanto si se trata de una edición de token, el atributo _acs será nulo.

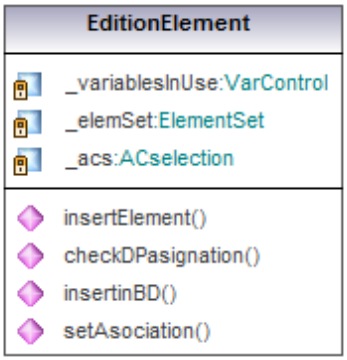


Ilustración 5.6: Clase de edición de elemento

Por otro lado, las operaciones *insertElement()* e *insertinBD()* que esta clase incluye no son más que métodos para simplificar su uso por parte del paquete vista ya que son implementadas por otras clases del controlador. Aún así está clase gestionará el tipo de elemento que se trata y restringirá o no las operaciones según su papel. También cuenta con métodos de checkeo como es *checkDPassignation()*, el cual verificará que todas las variables editadas hayan sido asignadas a alguna instancia de acto comunicativo, y el método para realizar estas asociaciones *setAsociation()*.

5.2.2.2 VarControl.

Esta clase pretende (Ilustración 5.7) englobar las operaciones de gestión de variables, para ello contiene una colección de nombres de variables. Ya que las variables editadas en ediciones de token se heredan en la primera edición de token o sub-token, es necesario llevar un control de estas. El método insertInherits() permite añadir una cierta cantidad de variables como variables heredadas, que son las que provengan de una edición de token anterior.

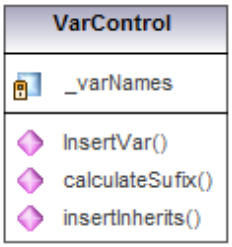


Ilustración 5.7: Control de variables

Por otro lado, se requiere calcular el sufijo de las variables según la instancia de token que se ha heredado. El método calculateSufix() lo que realiza es calcular el sufijo que se puede añadir a la instancia de token. Este valor depende de si hay variables

heredadas con anterioridad, por tanto comprobará por todas los nombres de variables locales, si al concatenar un sufijo X (que se irá comprobando) no crea conflicto con el nombre de alguna variable heredada. A continuación se ofrece un ejemplo para ilustrar este proceso:

1. Una edición de token (1) edita una frase como otro token (2).
2. En la edición de token (2) se anotan como variables las variables A y B. y completa su edición. Las variables según las ve la edición de token (1) se llaman A.1 y B.1 (ya que el sufijo calculado es 1).
3. La edición de token (1) contiene pues las variables heredadas A.1 y B.1 y además crea una variable local A. A la hora de calcular el sufijo, el valor 1 no es válido ya que concatenando la variable local A con el sufijo crea conflicto con la variable heredada A.1. Por tanto se incrementa el valor a 2, se concatena, y A.2 no crea conflicto con ninguna variable heredada.

Es necesario aclarar que cuando se concatena un sufijo, se concatena a todas sus variables, ya sean heredadas o no, por tanto, en el ejemplo anterior las variables para una edición superior serían A.1.2, B.1.2, A.2.

5.2.2.3 ACSelection.

Esta clase gestiona la selección de actos comunicativos. No tendrá uso si se trata de una edición de token, pero resulta indispensable si es de patrón o sub-patrón.

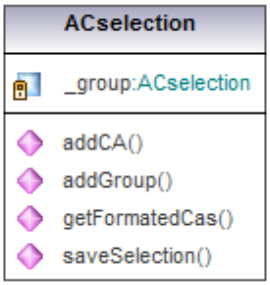


Ilustración 5.8: selección de CA.

Contiene un registro de instancias de actos comunicativos asignados mediante el método *addCA()*, que a partir de un acto comunicativo y sus pares categoría-valor seleccionados en la interfaz, crea una cadena de caracteres con la instancia y lo almacena en la colección.

Se ha decidido permitir la visualización de actos comunicativos asociados a otros elementos de nivel inferior a la edición, estos actos comunicativos son los llamados grupos (groups) y se visualizarán en otro color. Esta necesidad implica implementar una gestión similar y paralela a la de los actos comunicativos locales.

5.2.2.4 ElementSet

Esta clase gestiona el conjunto de elementos del que consta una edición. Contendrá un agregado de objetos de la clase CtrlPatternElement el que representará los elementos dentro de esta edición. Para ello implementa los métodos *addElement()* que añade elementos a la colección y *removeElement()* que elimina elementos de la colección. Este último se ha dejado como referencia para realizar la implementación en un futuro.

A su vez, realiza una gestión del orden de los elementos. En el momento de insertar un elemento, es muy dependiente su posición con el resto de elementos seleccionados, por tanto el método *addElement()* recibirá el orden del elemento con respecto a la edición en curso y reordenará según conveniencia la colección de elementos.

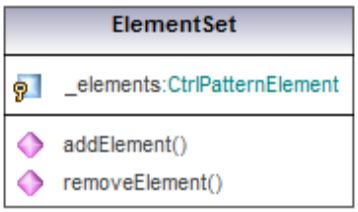


Ilustración 5.9:Clase ElemSet

5.2.2.5 CtrlPatternElement

Esta es la clase que representa un elemento de la edición. Se trata de una clase abstracta la cual al ser instanciada será, gracias al polomorfismo proporcionado por Java, un elemento concreto de los que aparecen en la Ilustración 5.10 que son:

- CtrlWord: Clase que representa un elemento de tipo palabra o *word*. Contiene la cadena de caracteres que corresponde a este elemento terminal.
- CtrlWildcard: Clase que representa los elementos comodines, contiene su tipo (definido o indefinido) y los valores en caso de que sea definido.
- CtrlSubPattern: representa un elemento de tipo sub-patrón. Contiene la referencia al patrón que forma parte de este elemento y la etiqueta dada para éste.

- CtrlToken: clase que representa un elemento de tipo token, contiene el identificador del token (de la colección de tokens), el nombre del token, su sufijo y su opcionalidad.

- CtrlParameter: Representa un elemento variable, contiene su nombre y su opcionalidad. El resto de datos los gestiona la clase VarControl.

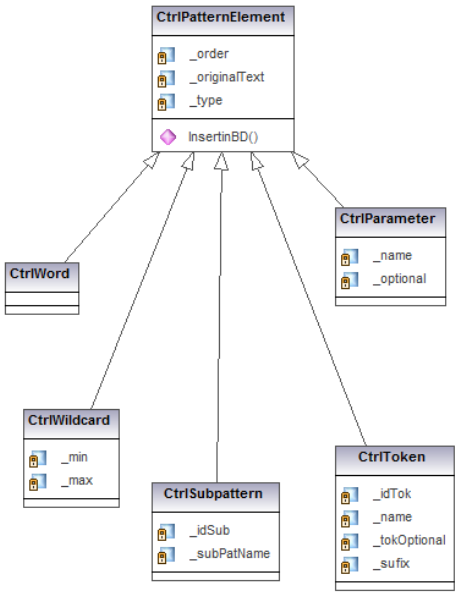


Ilustración 5.10: Gestión de elementos concretos.

Este diseño permite gestionar fácilmente cada uno de los elementos por separado ya que son independientes y cada uno de ellos posee ciertas características. Pero, a su vez, permite gestionarlos con independencia del tipo de elemento una vez editado. Por ello, l clase CtrlPatternElement posee el método *insertInBD()* el cual sobrescriben sus hijos y de esta forma cada elemento es responsable de su inserción en la base de datos.

5.2.3 NLIntegrator

Este componente se ha implementado como un módulo independiente del sistema para así permitir la reusabilidad en otras aplicaciones que requieran utilizar funcionalidad de interpretación y generación de lenguaje natural como son por el momento COGNOS.Dial o el Agente encargado de la generación en la arquitectura multi-agente (ver apartado 2.1).

Como se puede apreciar en la Ilustración 5.11, este componente se comunica con el procesador de lenguaje natural NLP y con el paquete modelo del sistema. Se encarga de

obtener los datos que existen en la base de datos y poblar la estructura en memoria del procesador, para que así este pueda sugerir un acto comunicativo en base a una expresión. Aunque el paquete NLP ha sido suministrado como caja negra, ha sido necesario realizar un estudio de su funcionamiento y estructuras para así poder cargar el conocimiento. Por tanto, se comentarán las interfaces de comunicación entre ambos componentes, el NLIntegrator y el NLP.

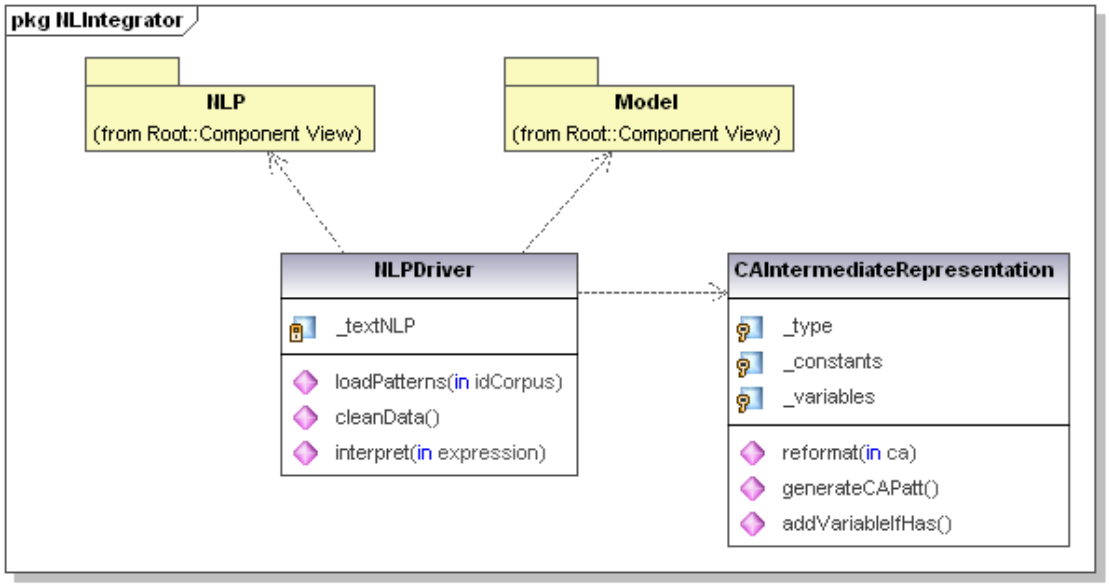


Ilustración 5.11: Componente de integración con el motor de lenguaje natural.

5.2.3.1 Análisis del procesador de lenguaje natural.

Se analizará en este apartado las estructuras de memoria y cómo se almacenan y se cargan, quedando el funcionamiento de interpretación y generación fuera de las competencias de este proyecto. Este componente utiliza las mismas estructuras de conocimiento tanto para interpretar como para generar, esto agiliza las necesidades en cuanto a carga de información se requiere.

Esta estructura en memoria se basa en una colección de patrones, cada uno de los cuales tiene asociado un conjunto de actos comunicativos en el formato en el cual se almacenan en la base de datos. Este formato (ver apartado 4.3.3) podría no ser el más adecuado, ya que con una correcta gestión se podría evitar el uso de las categorías ya que el orden de éstas para un determinado acto comunicativo identifica cada una de ellas. En un futuro se podrá realizar una integración completa.

En primer lugar, la clase más importante en cuanto a integración se refiere es TextNLP la cual contiene los “templates” que identifica cada patrón junto con su

conjunto de actos comunicativos. Esta clase pues contiene el siguiente método que se utiliza constantemente por cada patrón que se carga:

```
public void addTemplate(ExpressionPattern [] expressionPatternsOfTemplate,
List<CAPattern>caPatterns)
```

Un template está compuesto por objetos de tipo ExpressionPattern y de CAPattern y un template corresponde a un patrón editado para una expresión. La clase ExpressionPattern es una interfaz para abstraer todos los elementos de los que puede constar un patrón. A continuación se presentará la correspondencia entre los elementos en el procesador de lenguaje natural que implementan la interfaz ExpressionPattern y los elementos de la base de datos:

- **LeafPattern**: Referencia a los terminales de las gramáticas, es decir, las palabras. Contiene un String que representa dichas palabras.
- **JokerPattern**: Corresponde a los comodines de la gramática. Se ha modificado esta clase para que acepte comodines (en forma de expresiones regulares) infinitos o definidos, por tanto se dispondrá de comodines sin determinar (“`.*`”) y de comodines determinados ("`([\\w]+ +){"+ min + "," + max+"}`").
- **NodePattern**: Guarda relación con un patrón o sub-patrón. Contiene a su vez una colección de ExpressionPattern. Para este elemento han de encajar todos sus elementos, no existe variabilidad en ellos.
- **AdvanceNodePattern**: Se corresponde con un token. Contiene un array de dos dimensiones de objetos ExpressionPattern. La primera dimensión corresponde con los elementos que han de encajar como si de un objeto NodePattern se tratase. La segunda dimensión especifica las posibilidades que pueda tomar la primera. Por tanto serían las diferentes descripciones de un token.
- **VariablePattern**: Identifica los elementos variables editados. Contiene los mismos datos que un objeto de tipo LeafPattern pero el tratamiento de cara a la interpretación y generación es muy distinta ya que ha de encajar con las variables de la expresión o las variables de los actos comunicativos.

Una vez cargada la información anterior, se obtendrá el intérprete (o el generador según lo que se requiera) y se invocará al método expression2Ca de la clase intérprete que devolverá una lista de objetos CommunicativeAct que es la representación del motor de los actos comunicativos.

Por tanto el trabajo de este proyecto consiste en insertar en la estructura comentada anteriormente el conocimiento que se encuentre en la base de datos y hacer que el motor infiera sobre este. Por otro lado, el motor devuelve el CommunicativeAct en el cual están presentes pares categoría-valor. Existen valores modificados (si hay variables, serán sustituidas por el valor de esa variable en la expresión), y por tanto, es necesario cotejar el acto comunicativo obtenido con la base de datos.

5.2.3.2 NLPDriver y CAIntermediateRepresentation.

Estas son las clases implementadas para permitir la integración del motor de inferencia en el sistema. NLPDriver contiene una instancia de TextNLP y contiene tres métodos públicos: *loadPatterns()*, *interpret()* y *cleanData()*. El primero realiza la carga de información en las estructuras comentadas en el apartado anterior, se compone de dos métodos: loadPattern() y loadToken(). Si un elemento dentro de una edición es sub-patrón, será necesaria realizar llamadas recursivas a estos dos métodos. El segundo realiza la interpretación e intenta localizar el acto comunicativo y el tercero crea una nueva instancia de TextNLP realizando una llamada posterior al recolector de basura (*System.gc()*).

Es necesario incluir una gestión de actos comunicativos intermedia, ya que no existe asociación entre las variables editadas y las DPInstances del patrón. Por tanto, a medida que se cargan los patrones, se va modificando esta representación intermedia que contiene además del tipo una colección de parámetros variables (_variables) y una colección de parámetros constantes (_constants).

Al inicio de la carga se crea tantas instancias de CAIntermediateRepresentation como instancias de actos comunicativos tenga el patrón y todos sus parámetros constan en la lista de constantes (_constants). A medida que se va cargando los patrones y se van encontrando elementos de tipo variable, la categoría que contenga dicha variable será copiada en la colección de variables (_variables).

Una vez realizada la carga del patrón en las estructuras del NLP, este acto comunicativo intermedio contiene la estructura que necesita la clase CAPattern para crear un template, como se vio en el apartado anterior.

5.2.4 Modelo

El paquete modelo contiene la conexión a la base de datos y los mecanismos para acceder y almacenar todos los elementos de los que se compone el sistema. Para ello contiene cuatro componentes básicos: el componente de conexión, indispensable, y los tres componentes de datos agrupados según la parte de la base de datos utilizan para así permitir utilizarlos de manera independiente en diferentes herramientas.

5.2.5 Paquete de conexión.

Este paquete únicamente contiene una clase llamada DBConection que implementa el patrón de programación *Singleton*. Este diseño garantiza que sólo existirá en el sistema una instancia de esta clase ya que sólo se necesita una conexión a la base de datos por cada aplicación cliente. Por tanto esta clase contiene, como se aprecia en la Ilustración 5.12, un constructor privado *DBConection()* y un atributo del mismo tipo _conection. Las clases que requieran contener una instancia de esta clase realizarán una llamada al método *getInstance()*, que devuelve una nueva instancia si no ha sido creada (llamando localmente al constructor) o su atributo en caso de que ya haya sido instanciada con anterioridad.

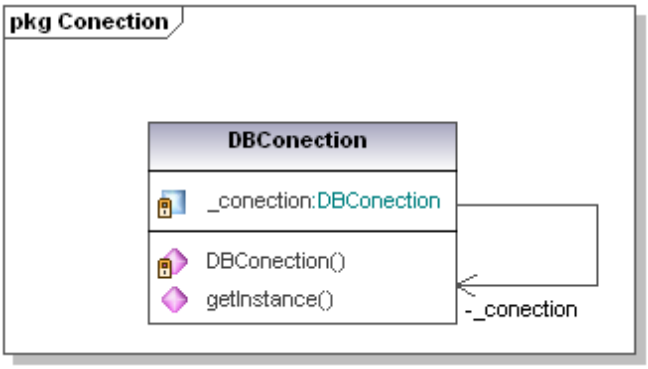


Ilustración 5.12: Implementación del paquete de conexión.

Esta clase contiene una conexión a la base de datos que proporciona el *driver* de conexión JDBC que es creada siempre que se invoque al constructor privado, es decir, una única vez por arranque de la aplicación. Esta conexión se crea de la siguiente forma:

```
con = DriverManager.getConnection (URL, username, password);
```

Donde URL es la cadena de host de conexión a la base de datos que en este caso es "jdbc:oracle:thin:@163.117.129.114:1521:DraftCognos"; En un futuro se permitirá desconectar y reconectar cambiando estos parámetros desde la interfaz de usuario, ya que las bases de datos no tienen por qué estar en la misma máquina ni llamarse de la misma forma. Es una línea futura muy deseable que será acometida a corto plazo.

Las sentencias SQL que se ejecutarán las elaborarán las clases de los paquetes de información ya que son muy dependientes del objeto y tipo de información que se manipule. Este componente ofrece la posibilidad de ejecución de una sentencia dada y la devolución de su resultado. Es pues, una capa intermedia entre los componentes concretos y la base de datos. El driver de conexión JDBC permite diferenciar dos operaciones de acceso y manipulación de datos: *executeUpdate()* y *executeQuery()*.

5.2.6 Paquetes de datos: CAModel, PatternModel e Identification

Estos tres paquetes son idénticos en cuanto a estructura y funcionalidad, el criterio de división responde a qué conjunto de tablas de la base de datos operan. Se ha decidido realizar este diseño para maximizar la modularidad de los componentes de la base de datos y, de esta forma, reutilizarlos en las diferentes herramientas que consta todo COGNOS. Por lo cual en este apartado se describirá la funcionalidad de estos componentes aunque se muestran todos en las ilustraciones que se encuentran en las páginas 76, 77 y 78.

Estos tres componentes se caracterizan por disponer de una clase general que sirve de interfaz al resto de componentes del sistema generalizando los métodos de consultas y modificaciones, siendo esta clase la que conoce la clase que elabora la consulta SQL. Por tanto, cuando le llega una solicitud a la clase general (BD_PatternModel, BD_Identification o BD_CAModel), ésta la procesa y solicita la acción a la clase especifica que corresponda, la cual elaborará la sentencia a partir de los parámetros y se comunicará con el componente de conexión para solicitar a la base de datos la transacción.

Estas clases concretas se encargan a su vez de obtener el resultado en forma de objeto del dato solicitado, transformando de esta forma el resultSet que devuelve el componente conexión en objetos o colecciones de objetos del tipo asociado a cada clase concreta según se aprecia en los diagramas.

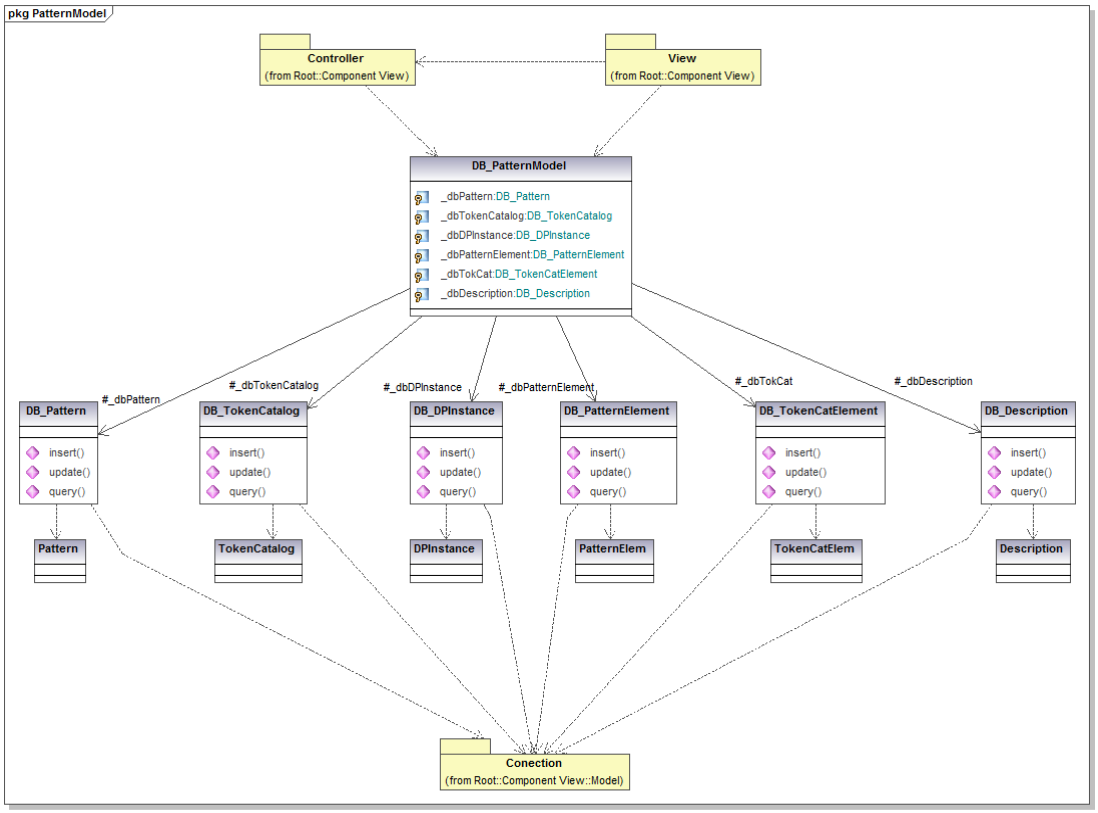


Ilustración 5.13: Implementación del componente PatternModel.

Se tomará como referencia en las explicaciones el componente PatternModel cuyo diagrama de clases es el de la Ilustración 5.13. Las ilustraciones que aparecen en los siguientes apartados son los diagramas de los componentes CAModel e Identification respectivamente.

5.2.6.1 DB_PatternModel

Esta clase contiene los objetos de las clases DB_* y aglutina todos los métodos que estas clases ofrecen implementando de esta forma una interfaz con el fin de minimizar el acoplamiento entre estos componentes. Para realizar esta interfaz, la clase DB_PatternModel dispone de instancias de las clases que realmente implementan los métodos, estas son (las antes llamadas DB_*) DB_CAsets, DB_CA, DB_AssignedCategory, DB_AssignedPair, DB_Constraints. Como se puede apreciar, cada una tiene del nombre de la entidad correspondiente en el diseño de la base de datos ya que cada una operará únicamente en su tabla homónima.

5.2.6.2 DB_CAsets, DB_CA, DB_AssignedCategory, DB_AssignedPair, DB_Constraints.

Estas clases son las que se comunican con el paquete de conexión a la base de datos y son las que elaboran todas las sentencias SQL relacionadas con cada entidad correspondiente a la clase y tratan el conjunto resultado encapsulándolo en objetos

creados para cada una de ella. La nomenclatura de estos sigue el mismo esquema, añadiendo el prefijo "DB_". De esta forma y por ejemplo, la clase DB_Pattern dispondrá de, entre otros, el siguiente método:

```
public Vector<Pattern> obtenerPatterns(int idCorpus)
```

Obtiene una colección de objetos Pattern (los objetos asociados a cada clase *DB_ **) a partir de la clave de corpus a los que estarán asociados. Para ello genera una consulta SQL para realizar esta operación.

En cada una de estas clases *DB_** se irán insertando tantas operaciones como necesidades de información fuera necesitando la herramienta ya que a priori no es posible conocer todas las consultas necesarias. Por el contrario sí que son conocidas las transacciones de actualización e inserción ya que la mayoría disponen de campos obligatorios. En caso de que algunos atributos admitan valores nulos, se realizará la sobreescritura de estos métodos para así facilitar las llamadas a las clases de otros paquetes.

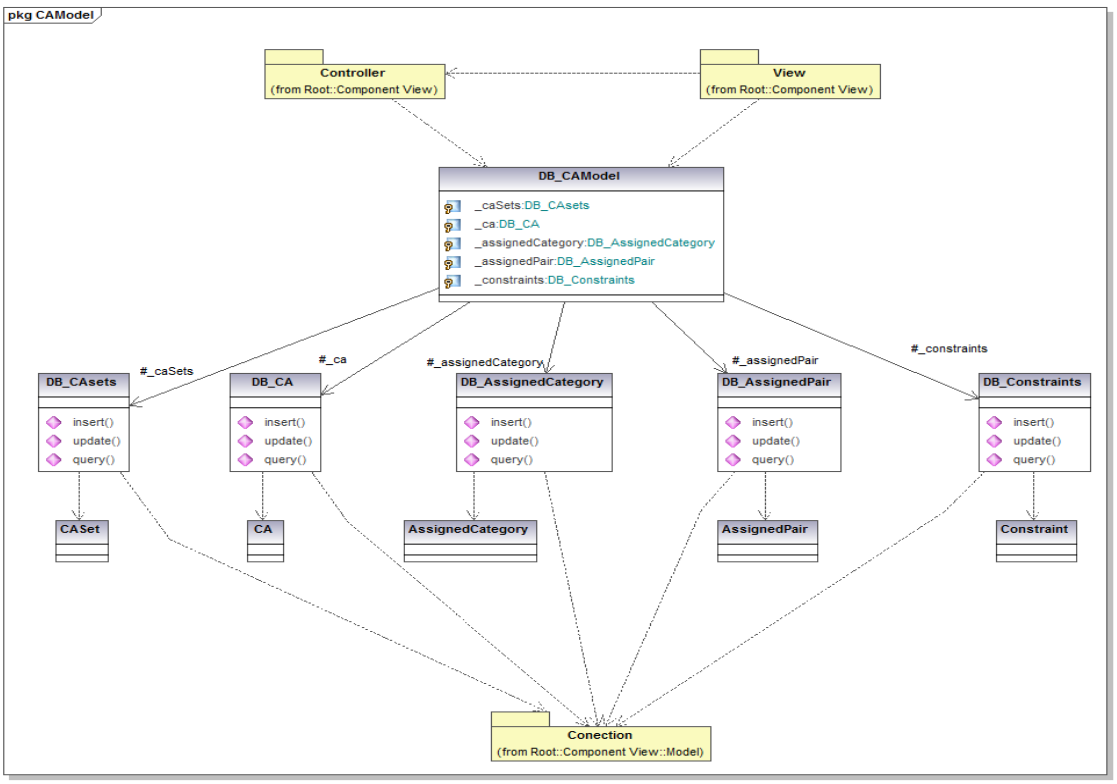


Ilustración 5.14: Implementación del componente CAModel.

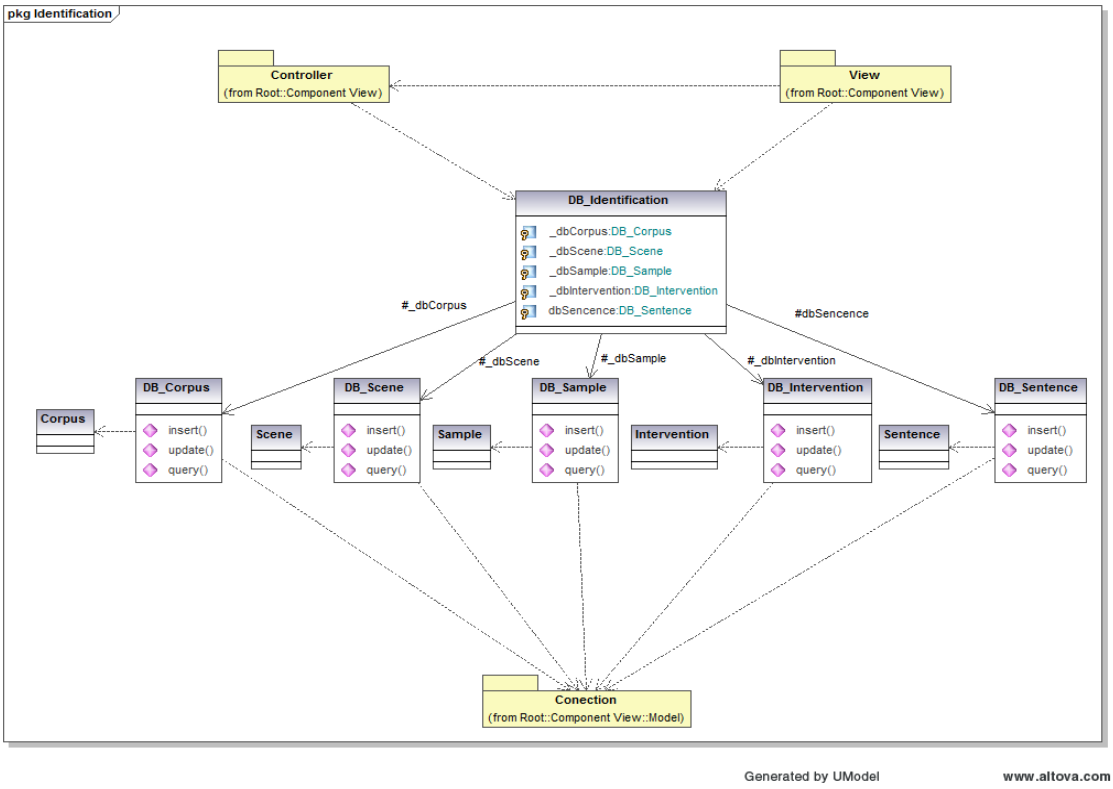


Ilustración 5.15: Implementación del componente Identification.

Almacenes de datos

En este apartado describirá la implementación de la base de datos junto con las soluciones para la semántica no recogida en el diseño conceptual del apartado 4.3. Se detallarán las decisiones de transformación tomadas para los elementos más complejos como son las jerarquías recursivas entre patrones y sub-patrones y entre tokens y sub-tokens.

El esquema relacional es el que muestra la Ilustración 5.16. Se aprecian lo que serán las tablas de las que dispone la base de datos Oracle, las claves primarias, claves ajenas y atributos. Este es el esquema global de la base de datos y lo conforman las tres partes del diseño pero para facilitar la comprensión de las explicaciones se dividirán éstas según sus tres apartados que son: apartado de identificación, de actos comunicativos y de patrones.

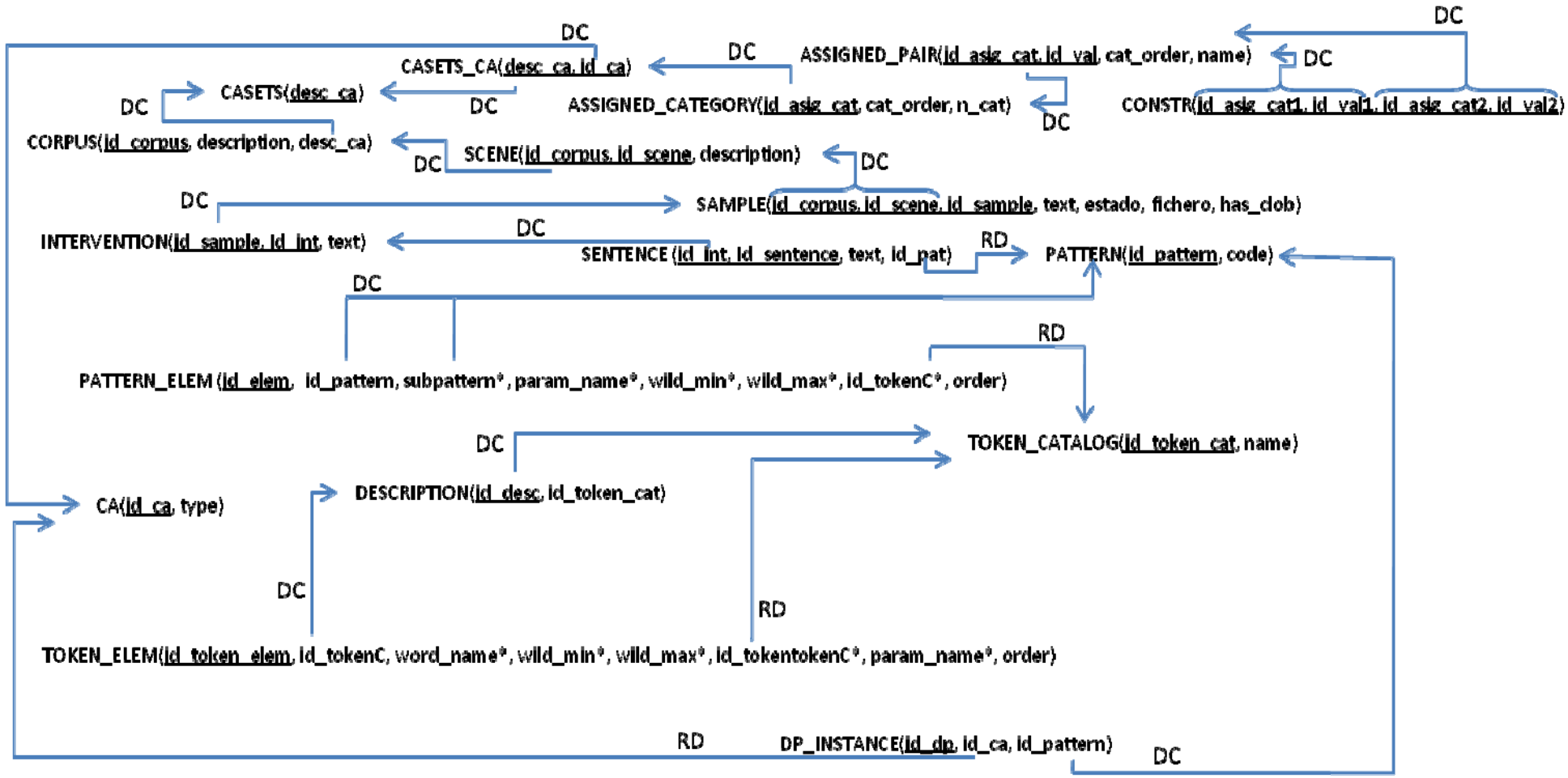


Ilustración 5.16: Grafo relacional

Se ha decidido que prácticamente todas las relaciones presentes en el diagrama que contengan una clave principal propia dispongan de una secuencia ordenada (con valor de comienzo cero) para dar valores a estas claves. Se planteó utilizar como clave el valor de ROWID que Oracle asigna por defecto a todas las tuplas de cada tabla y que toma como clave primaria si no se ha especificado ninguna. Pero debido a que algunos identificadores (como los de las tablas Corpus, Scene, Sample, Intervention y Sentence) son visibles al usuario, por aumentar la amigabilidad del sistema se ha decidido tomar la secuencia para realizar la asignación de claves. Para el resto de relaciones se usaron las secuencias para facilitar la depuración del sistema mientras se esté desarrollando.

5.3.1 Identificación.de Corpus

Las tablas que este apartado engloba son CORPUS, SCENE, SAMPLE, INTERVENTION y SENTENCE. Estas entidades (excepto CORPUS) son, en el diseño conceptual, débiles en identificación correlativamente. Esto se ha implementado mediante claves ajenas que forman parte de la clave principal de las entidades débiles, por lo cual la clave primaria de SCENE está compuesta por su clave propia y la clave de identificación de corpus. La clave de SAMPLE está compuesta por su clave propia y la clave de SCENE (es decir, la clave de CORPUS y la de SCENE) y así sucesivamente con INTERVENTION y SENTENCE.

Actualmente se ha omitido la posibilidad de que un patrón pueda ser asignado independientemente a un corpus o a una frase. Para permitir esto, se ha tomado la decisión de realizar una inserción automáticamente en el script de creación de la base de datos en las tablas de este apartado, por tanto se dispondrá siempre del identificador de CORPUS, SCENE, SAMPLE, INTERVENTION y SENTENCE con valor '0' reservado, y éste se representará como el "corpus general" a cuyas frases se les podrán asignar los patrones que no hayan sido identificados a un corpus en concreto.

Para dotar de lógica correlativa a las claves de estas tablas, se crea un disparador para SCENE, SAMPLE, INTERVENTION y SENTENCE en lugar de la secuencia que dispone el resto de tablas. Este disparador calcula la clave de estas tablas en función del valor máximo del último valor de la clave ajena. Esto se ha hecho de esta forma para que la clave de, por ejemplo, SAMPLE tenga un valor correlativo para el corpus 1 (aparezca como 1.1, 1.2, 1.3) y que para el corpus con clave 2 se reinicie la numeración.

A la hora de obtener estos datos para poblar el motor de inferencia, siempre se obtendrán los patrones asociados a un corpus determinado y los asociados al corpus general. Si no se especificase el corpus, se cargarán únicamente los patrones asociados al corpus general.

5.3.2 Actos comunicativos.

Este grupo lo forman las relaciones CA, CASETS, CASETS_CA, ASSIGNED_CATEGORY, ASSIGNED_PAIR, CONSTR. Como en el apartado anterior y con excepción de la relación CONSTR, el identificador propio de cada una es creado a través de una secuencia, aunque en estos casos este dato no sea visible al usuario.

Para realizar la transformación entre las entidades del esquema E/R del apartado 4.3 se han seguido las reglas básicas de transformación, como de esta forma se puede apreciar la tabla intermedia CASETS_CA que formaliza la relación N:M que forman en el diseño las entidades CA y CASets. Se ha decidido no crear una tabla nueva para la entidad Value, siendo sus atributos llevados a la tabla ASSIGNED_PAIR sin perder por tanto ápice de semántica ya que la clave primaria que figura en el diseño E/R de la entidad de mismo nombre pasa a estar formada por la clave llamada id_val junto con la clave ajena id_asig_cat.

Se ha creado un disparador para el valor 'order' de la tabla ASSIGNED_CATEGORY que asigne valores consecutivos para categorías asignadas al mismo acto comunicativo, empezando siempre por 0 si es la primera categoría asignada.

5.3.3 Apartado de patrones.

Este apartado engloba las relaciones que forman parte totalmente de la edición de gramáticas relajadas, estas son PATTERN, PATTERN_ELEM, TOKEN_ELEM, TOKEN_CATALOG, DESCRIPTION y DP_INSTANCE.

La decisión de implementación más significativa en este punto ha sido la transformación en una tabla las jerarquías que forman en el diseño los elementos de la edición de la gramática. Había que decidir si tener muchas tablas con pocos atributos o pocas tablas con tuplas que tengan muchos valores nulos. Al final la implementación ha sido la última ya que aporta mucha más velocidad el acceso a una única tabla con

atributos con pocos caracteres o nulos que obtener pocos datos de muchas tablas distinta.

Ya que se dispone de una única tabla para todos los elementos de patrón y otra para los elementos de token, es necesario introducir sendos chequeos para verificar que no se insertan datos de más o de menos en esas tablas, verificando pues el atributo tipo y dependiendo de su valor, comprobando la existencia de unos u otros atributos.

La tabla TOKEN_ELEM contiene una clave ajena no opcional la cual indica de qué descripción forman parte los elementos. Dispone de una clave ajena a la relación TOKEN_CATALOG (como PATTERN_ELEM) que indica cuando existe un elemento de tipo token, su referencia.

Se han implementado dos disparadores en esta sección. Ambos tienen la finalidad de calcular automáticamente el orden de un elemento en el momento de insertar en las tablas PATTERN_ELEM y TOKEN_ELEM. Estos dos disparadores contienen prácticamente el mismo código adaptado. Los disparadores realizan una consulta para obtener el número de orden máximo según el id del patrón (o token) que se está insertando. Si este número no es cero, se inserta como nuevo valor para el atributo order el obtenido más una unidad. Si por el contrario el valor es cero, se inserta como valor uno.

6 Verificación y validación

Este apartado tiene como objetivos realizar la verificación y la validación de la herramienta producida en este proyecto. La verificación tiene como finalidad comprobar que el sistema es correcto y funciona adecuadamente, la validación va más allá y se encarga de constatar que el sistema desarrollado cumple con los requisitos especificados. Esta fase aumenta la calidad de todo desarrollo software.

En el apartado 3.3.1 se especificaron unas pruebas que, a priori, comprueban todos y cada uno de los requisitos funcionales especificados en el apartado 3.1.2, en este apartado se llevaran a cabo estas pruebas, y si su resultado es válido se podría afirmar que el sistema desarrollado es válido y, por consiguiente, quedaría verificado.

El proyecto desarrollado tiene como producto final la adquisición de conocimiento de innumerables expresiones de lenguaje natural. Este conocimiento es almacenado en una base de datos, por tanto para verificar que la adquisición se ha completado (y por tanto la herramienta realiza su función) es necesario comprobar los datos almacenados en dicha base de datos. Se utilizará el programa de desarrollo SQL Developer proporcionado por Oracle que permite, entre otras muchas cosas, la monitorización del estado de la base de datos, así como comprobar su contenido.

Ejecución de las pruebas

En este apartado se especificarán los pasos seguidos para efectuar todas las pruebas expuestas en el apartado 3.3.1 partiendo desde la creación de la base de datos y realizando, una a una, las pruebas de validación. Se comentarán los resultados obtenidos en cada una de ellas, y las valoraciones pertinentes.

Para la instalación del sistema completo se requiere tanto la instalación y preparación de una base de datos en un servidor como la instalación del programa

editor. Estos pasos se toman como realizados ya que no aportan información sobre validación alguna. El sistema COGNOS.NL requiere información editada por otros componentes del paquete de herramientas, esta información es: conjunto de actos comunicativos genéricos editados mediante la herramienta COGNOS.CA y corpus anotados mediante la herramienta COGNOS.DIAL.

Resultados de las pruebas

Una vez ejecutadas todas las pruebas es necesario realizar un análisis de los resultados obtenidos. Con el fin de no repetir información, se incluye una tabla de resultados que hace referencia a las pruebas definidas en el apartado 3.3.1 y si éstas han sido satisfactorias o no. Si el resultado de una de las pruebas no fuera el deseado, se expondrá la causa de esto y las acciones que se deberán de llevar a cabo para paliar dicha deficiencia.

6.2.1 Tabla de resultados

En la siguiente tabla (Tabla 2) se aprecia los identificadores de las pruebas seguido de un icono que indica si éstas han sido satisfactorias o si, por el contrario, han devuelto un resultado no esperado o si no se han podido realizar. Por último se indicará la criticidad de cada una de las pruebas la cual viene determinada por el o los requisitos a los cuales verifica cada prueba.

Tanto la prueba número nueve como la trece no han sido satisfechas. La prueba nueve verifica que mediante algún mecanismo se pueda determinar el orden de las instancias de actos comunicativos asociadas a un patrón así como eliminarlas. La prueba número trece no es posible verificarla ya que no hay ningún mecanismo de inserción automática de intervenciones y expresiones por el momento. En el apartado siguiente se comentarán las deficiencias que acarrea el fallo de estas pruebas y las medidas que hay que tomar para minimizar su impacto.

Prueba	Resultado	Criticidad
P1	✓	Alta
P2	✓	Media
P3	✓	Alta
P4	✓	Alta
P5	✓	Alta
P6	✓	Alta
P7	✓	Alta
P8	✓	Media
P9	✗	Media
P10	✓	Media
P11	✓	Alta
P12	✓	Baja
P13	✗	Baja
P14	✓	Alta
P15	✓	Alta
P16	✓	Alta
P17	✓	Alta
P18	✓	Media
P19	✓	Alta

Tabla 2: Tabla de resultados

6.2.2 Acciones correctivas

Para definir acciones sobre pruebas defectuosas es necesario estudiar el motivo del fallo, los problemas que podría acarrear dicho error y por último concretar ya el plan de acción. Todo esto se detalla en este apartado.

La función que verifica la prueba número nueve no se ha llegado a implementar ya que fueron surgiendo requisitos con mayor criticidad. Se trata de un requisito de criticidad media ya que no es imprescindible para el correcto funcionamiento de la herramienta. Es más bien un requisito para facilitar la usabilidad y mejorar los efectos de un análisis mal realizado. Si el usuario edita los actos comunicativos concretos en el orden adecuado y éstos son correctos, este requisito no es necesario. Aún así, en sucesivas iteraciones está planeada su implementación.

La prueba número trece viene a verificar un requisito de criticidad baja, ya que es meramente una conexión con expresiones almacenadas mediante otra herramienta del toolkit COGNOS. No se pierde nada de funcionalidad ya que está disponible la posibilidad de escribir una expresión manualmente o incluso invocar a la herramienta COGNOS.NL desde COGNOS.Dial con una expresión concreta proveniente de un diálogo. En el momento en el que estas dos tablas de la base de datos sean pobladas se implementará este requisito.

7 Conclusiones y líneas futuras

En este apartado se expondrá las conclusiones a las que se ha llegado tras el periodo de realización de este proyecto así como las mejoras que se implementarán tanto a largo como a corto plazo o incluso ideas que mejorarían en gran medida la experiencia de usuario pero no se tiene planeado implementar debido a cuestiones que se explicarán en su correspondiente apartado.

Conclusiones

Los objetivos planteados han quedado cubiertos, permitiendo pues la obtención de conocimiento para ser usado por un procesador de lenguaje natural que pertenece al componente de interfaz de una arquitectura cognitiva contenida en un sistema de interacción natural. Los requisitos fundamentales han resultado satisfactorios aunque siempre quedan, como en todo proyecto del ámbito de la investigación, cuestiones por cubrir o incluso nuevas mejoras pensadas a posteriori. En el apartado 7.2 se describirán todos los detalles pendientes o posibles mejoras que han ido surgiendo a lo largo de todo el desarrollo.

También se ha conseguido diseñar una base de conocimiento e implementarla sobre un sistema gestor de base de datos. Será usada tanto por esta herramienta como por otras que forman parte del toolkit COGNOS. Se ha conseguido de esta forma integrar tres herramientas (COGNOS.NL, COGNOS.Dial, COGNOS.CA) que añaden y extraen conocimiento de esta base de datos.

La inclusión de esta herramienta suple una carencia importante en este ámbito. En el estado del arte (apartado 2.2) de este documento ya quedaba comentada la falta de herramientas con similar concepción y propósito. Una herramienta como la implementada agilizará el tedioso trabajo de la obtención de información sobre lenguaje natural haciendo que sucesivos proyectos de investigación que surjan se beneficien de

esta mejoría. De momento la herramienta ya se está usando para la anotación de corpus del grupo LaBDA y está en fase de evaluación.

Una de las principales complicaciones surgidas a lo largo de todo este tiempo ha sido llegar a comprender y estudiar unos términos tan ajenos a la informática como la multitud de conceptos lingüísticos que conlleva la herramienta. Por tanto, el conocimiento inicial se ha visto incrementado con valores de nuevas disciplinas. Toda formación académica recibida se puede ver reflejada en el producto final de este proyecto, desde las nociones sobre ingeniería del software hasta ingeniería del conocimiento, pasando por el diseño e implementación de bases de datos o programación en Java.

Líneas futuras

Todo proyecto contiene puntos en los que es necesario decidir sobre qué funciones implementar teniendo en cuenta el tiempo y los recursos disponibles. Toda funcionalidad que se pensó pero no ha sido posible realizar o las tecnologías actuales dificultan su desarrollo quedarán plasmadas en este apartado. Se han dividido, debido a la gran cantidad de ideas, en mejoras que se realizarán a corto plazo (ya que la herramienta sigue haciendo sus iteraciones de su ciclo de vida) y cuales a largo plazo.

Será necesario comentar que la herramienta será distribuida gratuitamente una vez esté completamente desarrollada. Para ello existen algunas de estas líneas futuras que si que se implementarán y otras que quedarán como posibles mejoras posteriores. Al publicar la herramienta se habilitará un mecanismo de recogida de opiniones, el cual servirá como fuente de nuevas líneas futuras.

7.2.1 Corto plazo

Sería recomendable que la herramienta fuese multilingüe: se tiene planeado publicar ya no solo COGNOS.NL sino todo el toolkit, y con el fin de abarcar el mayor número de usuarios posibles la herramienta debería de permitir al usuario escoger un cierto idioma. Se están barajando, de momento, dos idiomas, inglés y castellano. Si la herramienta tras su lanzamiento obtiene comentarios sobre la expansión a más idiomas, esto sería posible realizarlo.

Inicialmente, cuando se propuso el proyecto, se tenía pensado que únicamente iba a alimentar una base de conocimiento. A lo largo del desarrollo y ya con vistas a la distribución, se ve claramente la necesidad de incluir una pequeña interfaz para realizar conexiones a diferentes bases de datos. De esta forma se aumentará la movilidad de la herramienta.

Tomando como ejemplo la mayoría de aplicaciones con interfaz de usuario que existen, se ha pensado en incluir una barra de menú, en la cual se permitirá realizar todas las acciones posibles, así como incluir nuevas como la comentada en el párrafo anterior. A su vez, en esta barra se podrá incluir información sobre atajos de teclado para así agilizar aún más la anotación.

Es estrictamente necesario realizar algún cambio en la interfaz de usuario para permitir que cualquier pantalla pueda mostrar la aplicación. Hoy en día, al haber desarrollado la herramienta en vistas a uso personal, y como con la resolución de los equipos de trabajo se mostraba adecuadamente no se apreció este problema. La portabilidad a otros grupos de trabajo creó esta necesidad.

Será obligatorio en las siguientes iteraciones incluir la posibilidad de modificar el orden de las instancias de los actos comunicativos así como la de borrarlos. Esta funcionalidad forma parte de los primeros requisitos propuestos. No se implementó debido a su criticidad, pero será necesario incluirla antes del lanzamiento de la herramienta.

El diseño de la base de datos vista en el apartado 4.3 no recoge ningún dato sobre dominios de interacción y sobre cómo estos afectan a los corpus. La inclusión de esta nueva semántica permitirá unificar mejor la información referente a los corpus así como la reutilización de estos para sucesivos dominios que fueran surgiendo a lo largo de la fase de uso de la herramienta.

Será de utilidad para los usuarios finales de la herramienta que ésta incluyese un apartado de "ayuda" en el cual se muestre un resumen del manual de usuario y aclaraciones dependiendo de la acción que se pretende realizar.

7.2.2 Largo plazo

Una buena funcionalidad que es de agradecer en toda herramienta de edición general es la de deshacer cambios. Cualquier usuario desearía, tras un error de edición, remitirse a un punto inmediatamente anterior para reanudar su labor corrigiendo el error.

A medida que la herramienta sea usada para anotar diferentes dominios la información acerca de los tokens irá creciendo. La interfaz muestra todos los tokens editados previamente y esta amplia colección actualmente presenta una ordenación alfabética, pero sería recomendable implementar un mecanismo de búsqueda mediante la introducción de texto en la cual solo se mostrasen los tokens coincidentes con el texto introducido. Esto agilizaría en gran medida la edición de tokens en cuanto la información vaya creciendo.

Inicialmente se planteó la conexión de la herramienta con otra aplicación reconocedora de voz, que transmitiese una expresión a la herramienta. De esta forma el usuario quedaría eximido de introducir la expresión manualmente o, incluso de tener que transcribirla de cualquier otra fuente de audio.

Se ha propuesto, ya que la herramienta utiliza un motor de inferencia perteneciente a un procesador de lenguaje natural, que éste genere no tan solo una propuesta de actos comunicativos, sino un análisis completo para la expresión, dejando al usuario la posibilidad de decidir si el análisis propuesto es de su agrado o si por el contrario quiere anotar la expresión manualmente. Esta mejora, junto con la propuesta en el apartado anterior podría implicar en un futuro una anotación totalmente automática, siempre y cuando la base de conocimiento contenga suficientes datos.

Actualmente, los patrones que se editan (validados previamente) son guardados en la base de datos y utilizados por un procesador de lenguaje natural. No existe un mecanismo de acceso a estos patrones que no sea un cliente externo de bases de datos. Se ha pensado pues dotar a la herramienta de un mecanismo de recuperación de patrones antiguos e incluso de re-edición de estos.

Una mejora en términos de velocidad de generación por parte del motor de inferencia sería manipularlo para que, en lugar de inferir sobre estructuras previamente cargadas en memoria, obtenga los datos de la base de conocimiento directamente. De

esta forma si el motor obtiene los patrones a medida que infiere sobre cada uno de ellos, en el momento en el que encuentre un patrón, el motor parará su ciclo y no será necesario el acceso a toda la información de patrones de la base de datos.

8 Bibliografía

Austin, J. L. (1962). *How to Do Things With Words.* Oxford University press.

Calle, F. J. (2004). *Interacción natural mediante un procesamiento intencional: Modelo de hilos en diálogos.* Madrid.

Calle, J., Albacete, E., Sanchez, E., Del Valle, D., Rivero, J., & Cuadra, D. (2009). Cognos: a Natural Interaction Knowledge Management Toolkit. *NLBD.* Saarbrücken, Germany.

Dan Klein, C. D. (2003). Accurate Unlexicalized Parsing. 41st Meeting of the Association for Computational Linguistics.

Dan Klein, C. D. (2003). Fast Exact Inference with a Factored Model for Natural Language Parsing. Cambridge, MA: MIT press.

Edward Loper, S. B. (2002). NLTK: The Natural Language Toolkit.

IntegraTV-4all. (s.f.). Obtenido de www.tmtfactory.com/proyecto_integraTV-4all.asp

LaBDA. (s.f.). *Laboratorio de Bases de Datos Avanzadas*. Obtenido de http://basesdatos.uc3m.es/index.php?id=261

Langendoen, T. (2003). *Review: Natural Language Toolkit (NLTK) 1.2.* Obtenido de http://linguistlist.org/issues/14/14-3165.html

Mertz, D. (24 de Jun de 2004). *Charming Python: Get started with the Natural Language Toolkit*. Obtenido de http://www.ibm.com/developerworks/linux/library/l-cpnltk.html

Software Engineering Standards Comitee of the IEEE Computer Society. (25 de June de 1998). *IEEE Recommended Practice for Software Requirements Specifications.* Obtenido de http://www.kybele.etsii.urjc.es/docencia/IS4/extra/IEEE%20830-1998%20[ENG].pdf

Steven Bird, E. K. (s.f.). *NLTK book.* Obtenido de http://www.nltk.org/book

Sun. (s.f.). *NetBeans*. Obtenido de http://www.netbeans.org/

The Stanford Natural Language Processing Group. (s.f.). *The Stanford Parser: A statistical parser*. Obtenido de http://nlp.stanford.edu/software/lex-parser.shtml

The University of Sheffield. (s.f.). *4 pages brochure*. Obtenido de http://gate.ac.uk/sale/gate-flyer/2009/gate-flyer-4-page.pdf

The University of Sheffield. (n.d.). *GATE*. Retrieved from http://gate.ac.uk/

Anexo 1: Glosario.

Base de conocimiento: Almacén en el que se guardan ciertos datos que componen un conocimiento específico.

COGNOS KB: Base de conocimiento integrada del sistema COGNOS.

COGNOS Toolkit: Conjunto de herramientas de adquisición y gestión de conocimiento para la interacción natural.

Comodín: Se entiende por comodín aquel símbolo que sustituye a cualquier carácter o grupo de caracteres.

Corpus: Conjunto lo más extenso y ordenado posible de datos o textos científicos, literarios, etc., que pueden servir de base a una investigación.

Descripción de un token: Variante de formación de token con el mismo significado y función.

Entidad-relación: Herramienta para el modelado de datos de un sistema de información.

Escenario: Parte de las muestras de un corpus que tratan el mismo tema o guardan alguna relación entre ellas.

Expresión: Porción de intervención que no contiene ningún signo de punto y aparte ni ningún carácter de retorno de carro.

Inferencia: Acción y efecto de Sacar una consecuencia o deducir algo de otra cosa.

Interfaz de usuario: Mecanismo de comunicación usual entre los programas informáticos y sus usuarios.

Intervención: Ejecución de un turno por parte de un participante o secuencia consecutiva de expresiones dirigida a uno o a varios fines.

Latencia: suma de retardos temporales dentro de una red.

Librería de programación: conjunto de funciones externas que amplían las funciones propias del lenguaje de programación.

Máquina virtual de java: programa ejecutable en una plataforma específica capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial generado por el compilador del lenguaje Java.

Motor de inferencia: programa que, a partir de una carga de datos previos, es capaz de realizar inferencias sobre estos datos.

Patrón: Elemento principal en una gramática relajada.

Patrón de diseño: Diseño que busca solución a problemas de programación comunes.

Pragmática: Disciplina que estudia el lenguaje en su relación con los usuarios y las circunstancias de la comunicación.

Phyton: lenguaje de programación interpretado creado por Guido van Rossum en el año 1991.

Feedback (realimentación): En el ámbito de la ingeniería de software, este concepto se refiere a la acción de obtener opiniones y sugerencias de parte de terceros sobre una herramienta.

Sistema gestor de base de datos: Tipo de software muy específico dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan.

Token: Elemento de una gramática relajada que identifica a los terminales de esta. Está compuesto a su vez de diferentes elementos.

Toolkit: Conjunto de aplicaciones de un mismo sistema integrado.

Ventana emergente (pop-up): Tipo de interfaz de usuario que se activa generalmente sin que el usuario lo solicite cuando se necesita cierta información antes de realizar o continuar una acción.

Anexo 2: Acrónimos

API: ***A****pplication* ***P****rogramming* ***I****nterface*. Conjunto de funciones que proporciona un cierto lenguaje de programación al desarrollador.

EPSRC: Siglas de Engineering and Physical Sciences Research Council. Se trata de un consejo británico que da soporte y ayudas a la investigación y los estudios de post-grado en ingeniería y ciencias físicas.

MVC: ***M****odel-****V****iew-****C****ontroller*. Patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

PL/SQL: ***P****rocedural* ***L****anguage/****S****tructured* ***Q****uery* ***L****anguage***.** Lenguaje de programación incrustado en Oracle y PostgreSQL.

SGBD: ***S****istema de* ***G****estión de* ***B****ase de* ***D****atos.* Tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan.

UI: Siglas de *User Interface*, en castellano, Interfaz de usuario

XML: *E****x****tensible* ***M****arkup* ***L****anguage.* Metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C).

Anexo 3: Manual de instalación

La instalación de la herramienta de edición de gramáticas relajadas requiere, en primer lugar, tener copiada la carpeta Cognos_NL en el disco duro de la máquina del cliente (por ejemplo, en el directorio C:\). Puesto que Cognos.NL ha sido desarrollada en Java, es necesario tener su Entorno de Ejecución instalado (JRE). Por último, la máquina cliente ha de tener conexión a la red VPN de la Universidad Carlos III de Madrid ya que base de datos a la que se conecta el sistema se encuentra actualmente en un servidor del grupo LaBDA. A continuación se explicarán los pasos necesarios para instalar tanto el Entorno de Ejecución de Java como la Red Privada Virtual

Instalación del JRE

El Entorno de Ejecución de Java es posible descargarlo gratuitamente desde la siguiente página web: http://java.sun.com/javase/downloads/index.jsp. Únicamente es necesario descargar el archivo correspondiente al enlace *Java Runtime Environment (JRE) 6* (Ilustración AII-1) y seguir los pasos para su instalación.

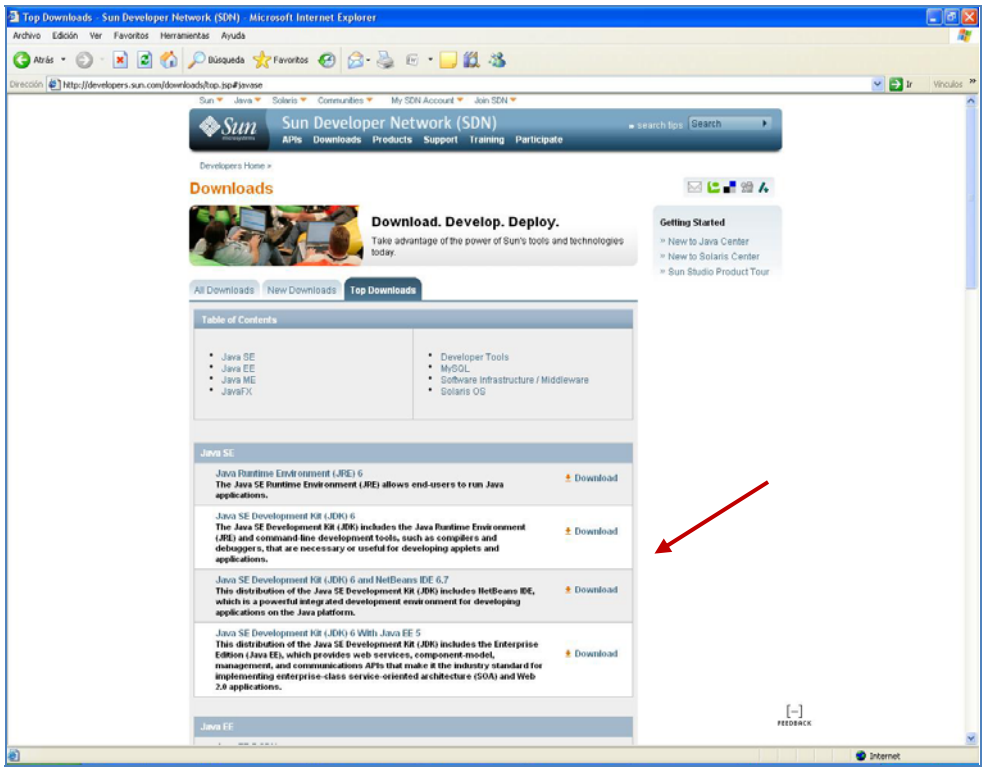


Ilustración AII-1: Descarga JRE

Configuración de la VPN

Para configurar la VPN en el sistema operativo y poder acceder a la Red Privada Virtual de la Universidad Carlos III, basta con seguir los pasos que se explican en el siguiente tutorial on-line: https://asyc.uc3m.es/index.php?Id=168 (Ilustración AII-2).

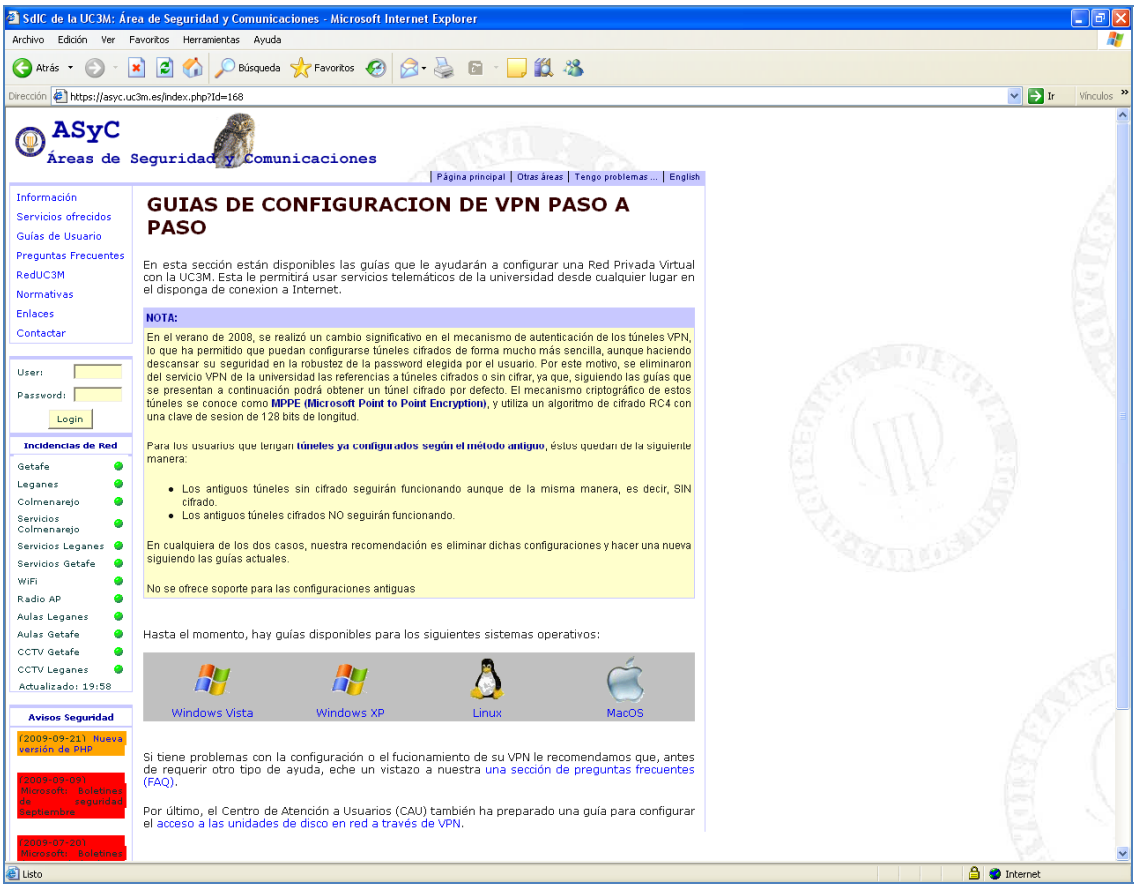


Ilustración AII-2: Configuración de la VPN

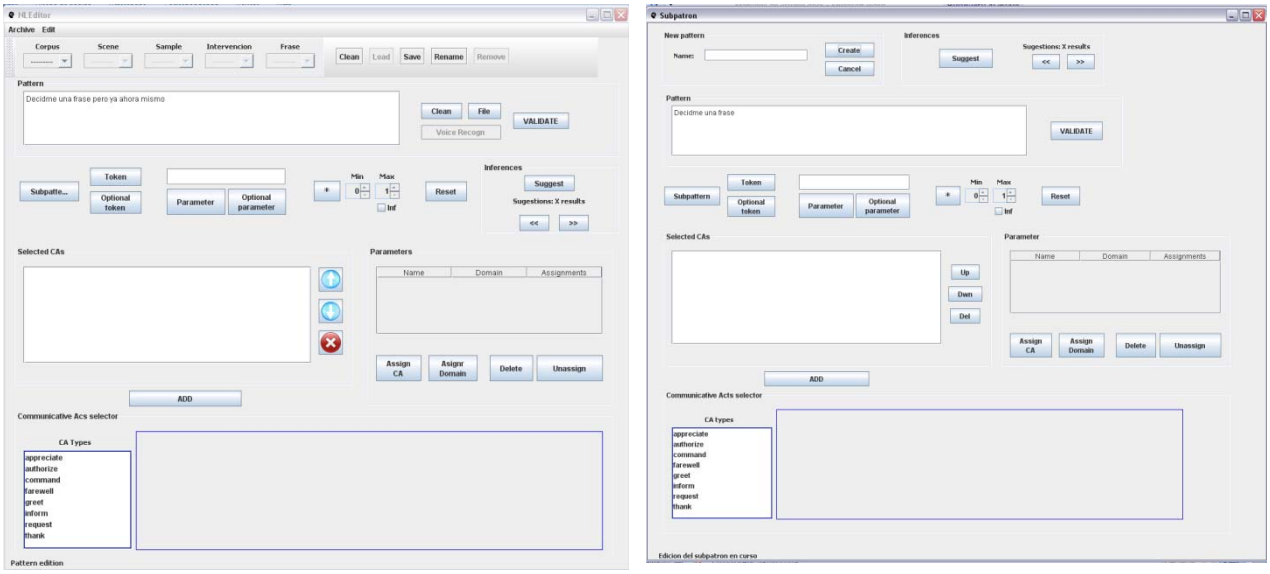
Una vez realizados estos pasos, para ejecutar la aplicación únicamente será necesario hacer doble click sobre el archivo cognos_nl.jar contenido en la carpeta Cognos_NL, ya que se ejecutará automáticamente con el JRE.

Anexo 4: Manual de usuario

Cognos.NL permite la edición de gramáticas relajadas a partir de expresiones en lenguaje natural, estas reglas serán almacenadas en la base de conocimiento para que otros sistemas lo consulten y operen con ello, como puede ser un motor de inferencia de lenguaje natural, o un agente externo. Dispone de dos interfaces principales: la edición de patrón y la edición de token.

Edición de patrón y sub-patrón

La interfaz de edición de patrón y de sub-patrón es muy similar y presenta contadas diferencias. La principal consiste en la eliminación de la identificación y en que en la edición de sub-patrón no se permite la edición de la expresión.



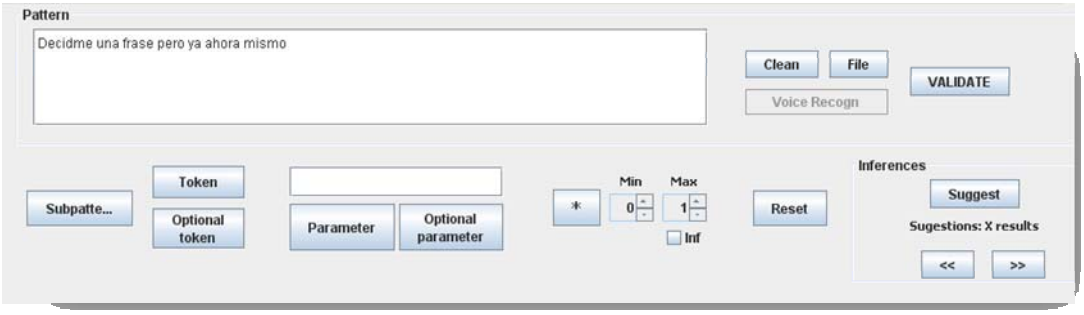
Edición de patrón principal vs edición de sub-patrón

La aplicación permite editar un patrón asociado a un corpus, un patrón asociado a una intervención que pertenezca a una muestra o una expresión introducida directamente y asociada al corpus global. Para todo ello se dispone de una barra de identificación en la cual se permite seleccionar un corpus, una frase a partir de su intervención, muestra escenario y corpus, o de no seleccionar nada.

Barra de herramientas principal

Además de la identificación, la barra principal contiene las operaciones generales que hacen referencia a la edición a ser: botón de limpiar [1] que borra la edición que pueda tener la expresión seleccionada. Botón de carga [2] que permite la recuperación de una expresión de la base de datos. Botón de guardado [3] que realiza el guardado de la edición, siempre y cuando ésta sea válida. Botón de renombrado [4] que permite cambiar la identificación, de esta forma se permite asignar una edición a otra expresión. Y por último, el botón de borrado [5], que elimina cualquier edición guardada para la frase seleccionada.

Dentro de la edición de un patrón se pueden identificar dos diferentes áreas, la edición de la expresión, en la cual se identificarán y editaran los elementos de la expresión, y la selección de actos comunicativos para asignarlos a la expresión.

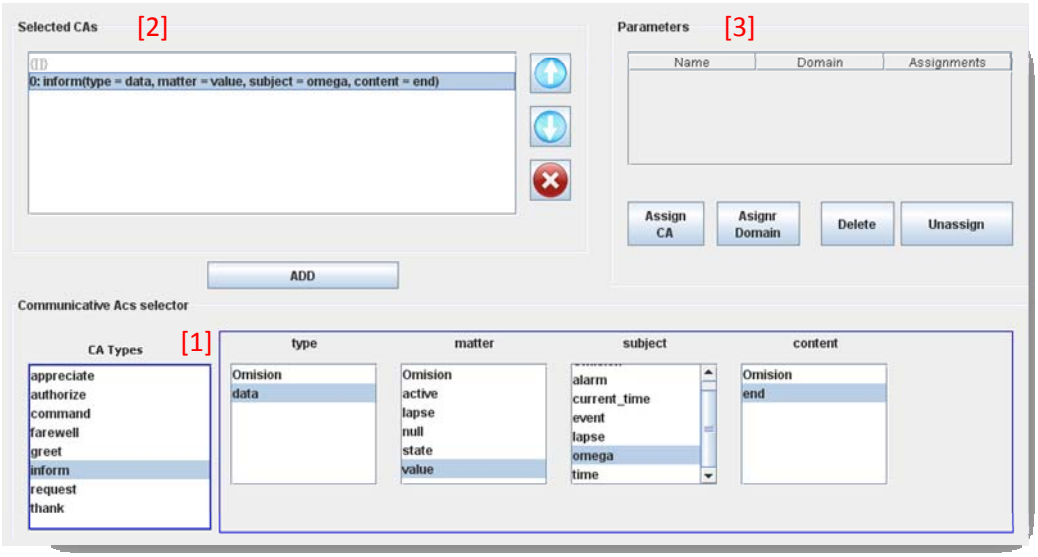


detalle de edición de elementos

Para la edición de un patrón se parte de una expresión en lenguaje natural. En ella se seleccionaran las palabras que se va a editar y a continuación se especificará que tipo de elemento es. Al lado de la caja de expresión, figuran los botones *limpiar*, *fichero* y *validar*. El botón limpiar, limpia la edición en curso, el botón fichero permite la adquisición de la expresión proveniente de un fichero de texto y el botón validar verifica si el patrón editado es válido. Si no es válido no se permite el guardado de este.

Si se selecciona una sub-expresión y se edita como sub-patrón, se abrirá la interfaz de edición de sub-patrón para esa expresión señalada. Si se edita como token o token opcional, se abrirá la interfaz de edición de token que se detallará más adelante. Si se edita como parámetro o parámetro opcional, es necesario especificar una etiqueta para éste y para editarlo como comodín es necesario especificar su cardinalidad máxima y mínima. Si la máxima es 0 o si se ha validado la casilla de inf, el comodín tendrá cardinalidad infinita, estos números se basan en cantidad de palabras.

Por último, aparece la sección de las inferencias, en la cual, en este momento sugiere mediante un motor de inferencia de lenguaje natural, un acto comunicativo aunque está pensado para que realice una edición automática de la expresión.



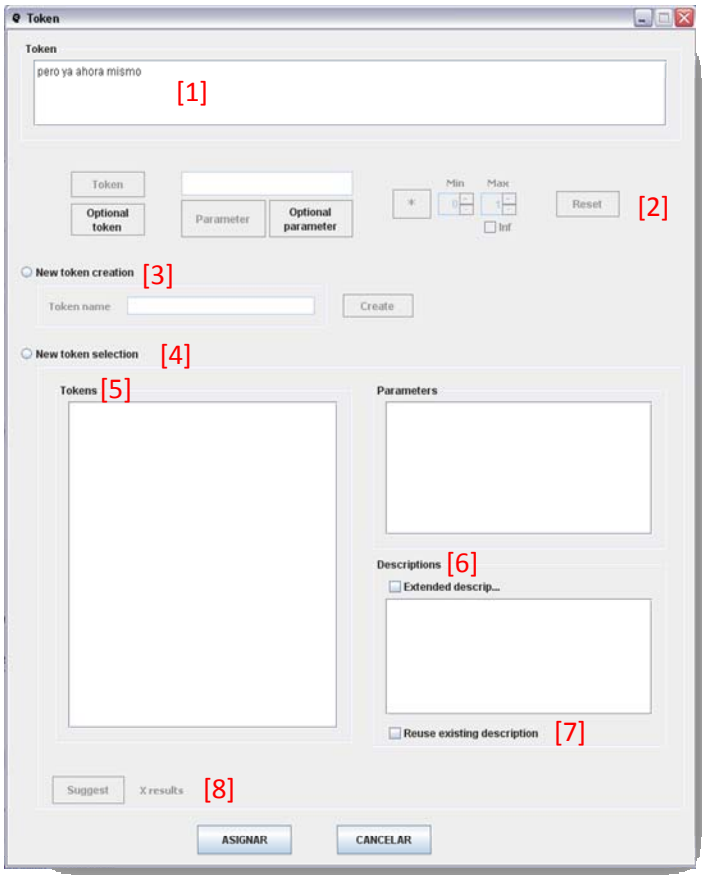
Detalle de la asignación de actos comunicativos

A un patrón o sub-patrón se le ha de asignar al menos un acto comunicativo para que la edición se considere válida. Para ello se deberá escoger el acto comunicativo en el selector de la parte inferior [1], para cada acto comunicativo aparecerán sus categorías y sus valores posibles. Los que se hayan editado con restricciones irán apareciendo según se vayan cumpliendo los requisitos. Una vez seleccionado el acto comunicativo adecuado, se asignará a la lista de grupo [2] mediante el botón *ADD*. Se permite editar el conjunto de actos comunicativos añadidos pudiendo eliminarlos del grupo y modificar su orden.

Si en la edición del patrón se ha insertado un parámetro, este aparecerá en la tabla *parameters* [3]. Para cada parámetro presente en esta tabla se podrá editar asignándolo al acto comunicativo seleccionado, editar su dominio (de momento una variable puede ser cantidad, fecha u hora, pero posteriormente se realizará por conceptos de ontología), eliminarlo de la edición y desasignarlo del acto comunicativo si procediese. Para que la edición sea válida, todas las variables han de estar asignadas al menos a un acto comunicativo.

Edición de token

Cuando se selecciona una o varias palabras y se editan como token, se abre una nueva interfaz de edición de token.



Interfaz de edición de token

En esta ventana se permite la edición de la expresión seleccionada anteriormente y se permite crear un nuevo token [3](asignándole para ello una etiqueta) o reutilizar uno ya existente [4] (reutilizándolo completamente o añadiendo un nuevo uso de ese token llamado descripción).

Para editar la expresión que se muestra en el cuadro de texto principal [1] se permite seleccionar el contenido. Los elementos de los que puede constar un token son: otros sub-tokens, parámetros, comodines o palabras. Todos los botines de edición se muestran bajo la expresión [2]. Los fragmentos de expresión que no hayan sido editados como sub-token, parámetro o comodín, automáticamente son considerados palabras.

Si se desea reutilizar un token existente, se activara la lista de tokens previos [5] con las etiquetas de todos los tokens presentes en la base de datos. Al seleccionar uno de estos, se mostrará en la lista de descripciones [6] todas las alternativas de definiciones para ese token. Es posible ver una descripción extendida, que expandirá la descripción de cualquier elemento.

Si no se activa la casilla de *reutilizar descripción existente* [7], pero se ha seleccionado un token, la edición que se ha realizado de la expresión se insertará como una nueva descripción para ese token, sin embargo, si se ha activado esta casilla, se reutilizará el token existente completamente y no se insertará nada en la base de datos. Este mecanismo es muy útil para analizar palabras muy comunes, por ejemplo, el token "saludo" constaría entre otras de la palabra "Hola", palabra que aparece muy comúnmente en castellano, por lo cual si se editan más expresiones con esta palabra, se reutilizaría completamente el token saludo, ya que ya existe una descripción idéntica.

Por último, existe un mecanismo de sugerencia [8] que seleccionará un token existente si hay alguna descripción que se adecúe a la expresión inicial.