UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

LENGUAJE NATURAL Y PLANEACIÓN AUTOMATIZADA PARA ROBOTS DE SERVICIO DOMÉSTICO

TESIS
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN CIENCIAS (COMPUTACIÓN)

PRESENTA:
SAMUEL SALVADOR VÁZQUEZ SÁNCHEZ

TUTOR:
DR. JESÚS SAVAGE CARMONA
FACULTAD DE INGENIERÍA

MÉXICO, D.F. FEBRERO 2015

Agradecimientos

A mis padres Lupita y Salvador, a mi hermano Benjamin y a todos mis amigos y compañeros.

Al Dr. Jesús Savage por asesorarme en la realización de esta tesis.

Al Dr. Sven Wachsmuth, de la Universidad de Bielefeld, en Alemania, por la orientación recibida durante mi visita a su laboratorio.

Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por el financiamiento de mis estudios de posgrado.

Al Programa de Apoyo a Proyectos de Investigación e Innovación Tecnológica (PAPIIT) por la ayuda económica al proyecto IN117612 "Robot de servicio para asistencia a adultos mayores y en sistemas hospitalarios".

Abstract

I present the design and implementation of a knowledge-based system that uses a symbolic representation of a microworld in order to translate statements in natural languaje into a sequence of high-level robotic activities. Three utterance types are considered: Statements about the world, questions about the state of the world and commands to a domestic robot.

The system has three main components:

- **Knowledge base**. Stores and retrieves symbolic information about the world.
- **Natural language processor**. Translates a natural language statement into a formal meaning expression.
- **Action planner**. Uses a meaning expression and the information about the world to produce a high-level plan that the robot must execute.

The main purpose of the system is to install it in the robot *Justina*, developed in the *Biorobotics Laboratory* (UNAM), and use it in competitions such as *RockIn@home* y *Robocup@home*. In particular, the system was used in the task *Speech Understanding* of the competition *RockIn* 2014, in which the team placed second. In addition, using a third-party action planner, the system was used to command the robot *Justina* to arrange cubes, using natural language.

I also used the system to process the 2014 SEMEVAL corpus in the task 6 *Supervised Semantic Parsing of Robotic Spatial Commands*, which consists of more than 3,400 natural language commands to arrange geometric figures.

Resumen

En esta tesis presento el diseño e implementación de un sistema que usa una representación simbólica del mundo para traducir oraciones en un lenguaje natural controlado a una secuencia de acciones de alto nivel. En otras palabras, planteo un sistema basado en conocimiento que interpreta oraciones de un fragmento de un lenguaje natural para manipular la representación de un micromundo. Se consideran tres tipos de oraciones: Declaraciones sobre el estado del mundo, preguntas sobre el estado del mundo y órdenes a un robot de servicio doméstico.

El intérprete tiene tres principales componentes:

- **Base de conocimiento**. Almacena y extrae información simbólica del modelo del mundo.
- **Procesador de lenguaje natural**. Traduce una oración en lenguaje natural a una expresión formal de significado.
- **Planeador de acciones**. Usa una expresión formal de significado y el estado del mundo para entregar una secuencia de acciones que el robot debe ejecutar.

El propósito principal del sistema es usarlo en las competencias de robótica *RockIn@home* y *Robocup@home* al integrarlo al software de control del robot *Justina*, desarrollado en el *Laboratorio de Biorobótica* en el *Posgrado de Ingeniería* en la UNAM. En particular, colaboré en la preparación de la prueba *Speech Understanding* en la edición 2014 de la competencia *RockIn*, donde se obtuvo el segundo lugar. Además, usando un planeador automático de una tercera parte, se logró integrar el intérprete de lenguaje natural descrito en este trabajo para comandar al robot *Justina* a ejecutar actividades de manipulación de bloques de colores.

También usé el sistema para procesar el corpus SEMEVAL 2014 en la tarea *Supervised Semantic Parsing of Robotic Spatial Commands*, que consta de más de 3,400 órdenes en lenguaje natural para un robot manipulador de prismas.

Contenido

Capítulo 1

Introducción

Comunicarse mediante un lenguaje natural y poder planear acciones antes de ejecutarlas son dos cualidades que necesitan los robots de servicio para poder integrarse en actividades domésticas cooperativas con humanos. En la medida en que los robots aumenten sus habilidades en representación de conocimiento, procesamiento de lenguajes naturales y planeación automatizada, se podrán crear interfaces de usuario más amigables que permitan una cooperación efectiva y precisa entre humanos y máquinas. Los lenguajes naturales son el medio de comunicación más efectivo para establecer interacciones verbales entre humanos y máquinas porque son fáciles de entender por los humanos y pueden expresar ideas e intensiones complejas. Sin embargo, los lenguajes naturales son ambiguos y su análisis se considera computacionalmente poco eficiente [1].

Un lenguaje natural controlado es un fragmento de un lenguaje natural, restringido en la complejidad gramatical y el tamaño del vocabulario. En esta tesis presento el desarrollo de un sistema que usa una representación simbólica del mundo para traducir oraciones en un lenguaje natural controlado a una representación formal del significado. Cuando el enunciado de entrada es una orden a la máquina, un planeador automático genera una secuencia de acciones de alto nivel que corresponden a los comportamientos que debe ejecutar el robot como respuesta a la actividad solicitada. El contexto en que se desarrolla el proyecto es el de un robot humanoide de servicio doméstico para ayudar a adultos mayores. No obstante, este trabajo puede ser adaptado a otro tipo de micromundos similares donde se desee controlar una máquina con oraciones en lenguaje natural.

1.1 Descripción del problema

Se cree que en un futuro los robots serán capaces de atender necesidades domésticas cotidianas. A pesar de que tales robots aún están lejos de ser un producto para el consumo masivo, iniciativas como las competencias internacionales *Robocup@home* y *RockIn@home* promueven la actividad académica en esa dirección. Dichas competencias establecen lineamientos específicos sobre pruebas que robots humanoides deben cumplir en escenarios cotidianos. Este trabajo tiene como contexto los lineamientos de tales competencias y las capacidades de los robots que participan en ella, en especial del robot *Justina*, desarrollado en el *Laboratorio de Biorobótica* del Posgrado de la Facultad de Ingeniería de la Universidad Nacional Autónoma de México.

Consideré tres tipos de interacciones verbales que el usuario puede tener con el sistema:

- Declarar un hecho.
- Hacer una pregunta sobre el estado del mundo.
- Indicar una orden a un robot doméstico.

El problema central de este trabajo es el diseño de un intérprete que traduce oraciones en lenguaje natural a comportamientos del robot, desde adquirir nuevo conocimiento hasta redistribuir objetos en distintas mesas. En la figura 1.1 se muestra una descripción general del problema a resolver.

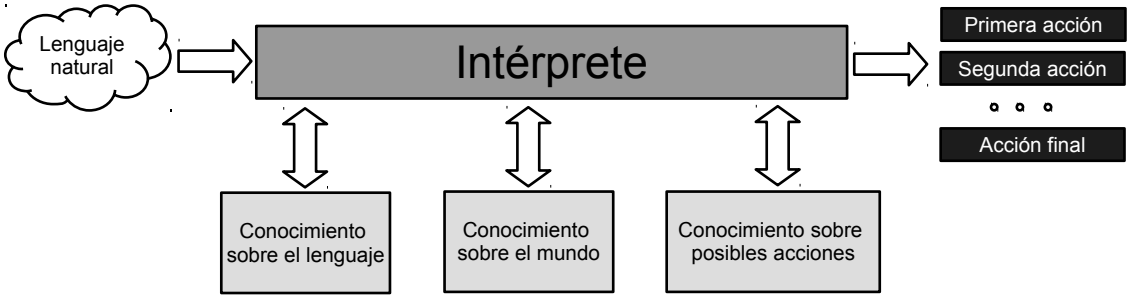


Figura 1.1 Descripción gráfica del problema a resolver.

El intérprete tiene tres principales componentes:

- **Base del conocimiento**. Almacena y extrae información simbólica sobre el modelo del mundo en el sistema.

- **Procesador de lenguaje natural**. Traduce una oración en lenguaje natural a una expresión formal de significado.

- **Planeador de acciones**. Usa una expresión formal de significado y el estado del mundo para entregar una secuencia de acciones que el robot debe seguir para cumplir la orden solicitada.

1.2 Trabajos relacionados

El uso de lenguaje para ingresar y obtener información de un sistema computacional y para controlar máquinas se ha planteado y explicado desde distintas perspectivas, las cuales tienen distintos compromisos dependiendo del problema que resuelven.

Los proyectos presentados como estudios de interacciones humano-robot basadas en lenguaje varían en contexto desde videojuegos con misiones cooperativas, galerías de arte [11], comunicación de ubicaciones y direcciones [12] [13], robots de trabajo [2] y robots de servicio doméstico [17] [15] [16]. Estos proyectos frecuentemente se plantean en términos de las interacciones humano-robot y buscan integrar conocimiento sobre inteligencia artificial, lingüística y psicología.

El uso de ontologías para almacenar conocimiento relativo al micromundo de robots es frecuente y existen trabajos directamente relacionados con robots domésticos [15] [17], para describir robots guías en exhibiciones de arte [11] y han servido incluso para inferir la intención de un usuario humano con el que se comparte un espacio de trabajo [2]. En procesamiento de lenguaje, las ontologías se han usado para almacenar información lingüística y del modelo del mundo simultáneamente, como en *Generalized Upper Model* (GUM) y en *Descriptive Ontology for Linguistic and Cognitive Engineering* (DOLCE). También se han usado para almacenar información de la web semántica como en *General Ontology for linguistic description* (GOLD) [3].

Otros grupos de trabajo han explicado la forma en que fragmentos de un lenguaje natural sirven para expresar hechos, preguntas y reglas en sistemas basados en conocimiento, como *Atempto Controlled English* [4], *Computer-Processable Language* [5] y *Processable English* [6], e incluso se ha usado un lenguaje natural controlado como un lenguaje de programación de propósito general [14]. Por otro lado, se ha argumentado que es posible interpretar lenguaje natural con muy pocas restricciones, incluyendo errores involuntarios, si se mantiene limitado el tipo de interacciones humano-robot, como al indicar direcciones y ubicaciones [12]. También existe trabajo sobre el acceso a bases de datos usando lenguaje natural como el sistema PRECISE [8], la mayor parte de estos sistemas se basan en lógica y algún tipo de información sintáctica que enlaza el nombre de las tablas y campos con palabras en el lenguaje, véase [10] para una semblanza histórica en interfaces de lenguaje natural a bases de datos.

En planeación automatizada para robots de servicio doméstico, se han usado máquinas de estado jerárquicas, planeadores de redes jerárquicas de tareas [15] y lenguajes de programación diseñados para ambientes dinámicos como GOLOG [16] y SitLog [18]. El robot *Justina*, del *Laboratorio de Biorobótica* del *Posgrado de Ingeniería* en la *UNAM*, actualmente usa máquinas de estados jerárquicas para describir el comportamiento de alto nivel. El robot *Golem II+*, desarrollado en el *Instituto de Investigaciones en Matemáticas Aplicadas y Sistemas* en la *UNAM*, usa el lenguaje SitLog [1] [30].

1.3 Contribución de esta tesis

Desde un par de décadas atrás han surgido propuestas de especificación de lenguajes naturales controlados que sirven como un lenguaje de alto nivel comunicando una persona con algún tipo de sistema basado en conocimiento. La especificación de estos lenguajes es relevante debido a que usan lenguajes más sencillos de entender para personas sin preparación técnica, en contraste con los lenguajes formales. Además, al procesar un lenguaje natural controlado permite que se pueda traducir automática y determinísticamente a un lenguaje formal objetivo donde existan mecanismos automáticos de razonamiento. Varias especificaciones de lenguajes naturales controlados se presentan en [4], [5] y [7] pero la especificación de significado que presentan falla en incluir mecanismos de consulta a módulos externos, como pueden ser agentes de percepción o bases de conocimiento.

En este trabajo propongo una forma de especificar fuentes externas de consulta cuando los roles semánticos necesarios no están presentes en el enunciado. La especificación de significado se expresa en patrones que constan de una lista de las descripciones de los roles semánticos que pueden ocurrir en la frase y de una expresión que al evaluarla conforma la expresión de significado de la oración de entrada. La configuración del sistema implica trabajo manual por parte del diseñador de la aplicación para especificar el lenguaje aceptado pero intenta minimizar el uso de información sintáctica y ofrecer un control directo sobre el significado deseado de cada categoría de oraciones.

Además, este trabajo se adaptó para usarse en competencias de robótica como *RockIn@home* y *Robocup@home* donde se espera que provea al robot mecanismos de deliberación más robustos y generales para establecer interacciones más naturales y consistentes con el usuario.

[1] SitLog es un lenguaje lógico, declarativo y orientado a situaciones para programar tareas de robots de servicio doméstico que usa el modelo computacional de redes de transición recursivas funcionales (F-RTN). Estas tareas se describen como protocolos genéricos llamados modelos de dialogo (DMs), cada DM está compuesto de un identificador, un conjunto de situaciones y un conjunto de variables locales. En el proceso de interpretación, instancias de DM y situaciones son creadas y ejecutadas dinámicamente, expandiendo un grafo que corresponde a la estructura de la tarea [18].

1.4 Organización de los capítulos

En el capítulo 2 *Planteamiento* se describe el tipo de interacciones que se esperan resolver para explicar el alcance del trabajo presentado. Luego se da un resumen de la propuesta de solución presentando las ideas medulares de diseño.

En el capítulo 3 *Representación de conocimiento simbólico* se da una breve explicación de los conceptos teóricos sobre representación del conocimiento y se describe la base de conocimiento diseñada.

En el capítulo 4 *Formalización del lenguaje* se explican algunos conceptos lingüísticos relevantes para la tesis y se presentan los algoritmos que integran el intérprete de lenguaje natural.

En el capítulo 5 *Planeación automática* se incluye una breve explicación de planeación con redes jerárquicas de tareas y se describe la manera de especificar *planes*.

En el capítulo 6 *Implementación y codificación de micromundos* se mencionan detalles sobre la implementación del sistema y la codificación del dominio del micromundo de un robot de servicio doméstico.

En el capítulo 7 *Evaluación* se analizan las capacidades del sistema. La evaluación de lenguaje incluye resultados de pruebas de la competencia *RockIn@home* y con el corpus SEMEVAL tarea 6. La evaluación del planeador de acciones incluye pruebas simuladas similares a la competencia *Robocup@home* y pruebas simples con un robot real.

En el capítulo 8 *Conclusiones y trabajo futuro* se dan las observaciones y conclusiones finales sobre lo presentado, así como los aspectos que aún pueden mejorarse o generalizarse. Además, se menciona una opinión respecto a los retos al futuro en la interacción verbal humano-robot.

Capítulo 2

Planteamiento

Un asistente doméstico debe ejecutar órdenes con precisión y robustez. Para ello se requiere proveer al robot con habilidades complejas como reconocer rostros, objetos, desplazarse en lugares conocidos y desconocidos, manipular objetos con destreza entre muchas otras capacidades que son en sí mismas campos de investigación interdisiplinaria. Sin embargo, estas máquinas no serán atractivas hasta que sean lo suficientemente confiables y fáciles de usar por cualquier persona. Por ello, el procesamiento de lenguaje natural y la planeación en las decisiones de alto nivel en un robot de este tipo representan problemas centrales en el desarrollo de los robots domésticos.

Muchas de las decisiones de diseño respecto al tipo de conocimiento requerido, el espectro del lenguaje que se desea formalizar y la descripción de las tareas de planeación dependen del grado de detalle con el que se represente el mundo. En este trabajo utilizo únicamente información simbólica que representa una abstracción del mundo adecuada para un robot humanoide que forma parte de la asistencia doméstica en un hogar. De forma similar, diseñé el planeador de acciones para enumerar comandos de alto nivel que son ejecutados por otros agentes de un nivel de abstracción menor dentro de la pila de software que controla al robot, como son los agentes de localización, desplazamiento, reconocimiento de imágenes, reconocimiento de voz, etc.

Para determinar el alcance del estudio presentado se consideró que el sistema debe ser capaz de:

- Almacenar nuevas palabras y asociaciones entre conceptos mediante interacciones verbales. Conforme se adquiera nuevo conocimiento, se debe integrar inmediatamente a los procesos de comprensión de lenguaje y planeación.
- Comunicar frases de confirmación y respuestas en lenguaje natural.
- Interpretar y resolver órdenes del tipo que se presentan en la prueba *EGPSR* de la competencia *Robocup@home*.

En el resto de la sección se describen los escenarios prototipo en que el sistema debe funcionar y se detalla la propuesta técnica para resolver las interacciones planteadas.

2.1 Interacciones esperadas

Plantear el lenguaje que debe comprender un robot lleva a definir primero el tipo de información que debe conocer. Por esta razón, el diseño de solución debe favorecer la implementación de interfaces de lenguaje que comprenda una gran cantidad de paráfrasis de oraciones en un dominio restringido. En contraste con otros campos de procesamiento de lenguaje natural que se concentran en textos de dominio abierto, el planteamiento de esta tesis se concentra en explotar el conocimiento que el diseñador de la aplicación tiene sobre el dominio y el contexto en que se da la interacción con el usuario. El robot no necesita saber cosas que no le sirven para cumplir su propósito. A continuación se presentan algunas ideas sobre el propósito que debe cumplir un robot de servicio doméstico. Se propone el siguiente escenario hipotético:

Un robot humanoide está en un departamento donde es parte de la asistencia doméstica. Entre múltiples dispositivos mantienen actualizada la información del estado del mundo aunque existen parámetros del mundo que no se pueden conocer en tiempo real. El robot en cuestión puede trasladarse entre cuartos y alinearse en cualquier posición relativa a un objeto. Posee al menos un brazo manipulador que puede sostener objetos pequeños y ligeros, puede también percibir rostros, reconocer objetos conocidos como bebidas o muebles, evadir obstáculos en movimiento, también puede escuchar y emitir voz

El robot está en la cocina y una persona llega a hablarle.

- ¿Dónde están mis llaves?

El robot reconoce el rostro de la persona y sabe que posee las llaves de un auto y las llaves de una casa. El robot responde:

- Las llaves del auto están en la mesa que está en el cuarto principal; las llaves de la casa están en la mesa cerca al sillón.

- Hay unos artículos en la cocina. Guárdalos.

El robot sabe que es probable que las compras estén en la mesa de la cocina. Se alinea a ella y ejecuta rutinas de reconocimiento de objetos para identificarlos. Observa cajas de leche, fruta, latas, etc. Toma cada uno de los objetos y los guarda en el refrigerador o en la alacena según corresponda.

Si bien el escenario es aún modesto, es un punto de partida para ejemplificar cómo el lenguaje y las situaciones comunes en nuestra vida conllevan una gran cantidad de

información implícita en el contexto.

Para resolver la pregunta *¿Dónde están mis llaves?* el robot debe tener codificado cómo ligar un rostro de una persona con su nombre, saber que ese nombre representa un humano y que un humano puede poseer cierto tipo de objetos. Además, debe saber que una llave es un objeto físico que puede tener una ubicación espacial única y que esa ubicación puede expresarse en términos relativos a otros objetos que el robot también conoce. Debe saber que las relaciones entre objetos se pueden propagar por transitividad por lo que si unas llaves están en la mesa en el cuarto principal, entonces esas llaves también están en el cuarto principal. El robot debe saber que existen instancias de llaves presentes en el departamento que pertenecen al humano que le está hablando y debe existir un mecanismo que le permita extraer todos los atributos explícitos o implícitos de esos objetos.

Para resolver las oraciones *Hay artículos en la cocina* y *Guárdalos* el robot debe tener almacenado que incluye la categoría de objetos *artículos* y qué características los diferencian de otros objetos. Además para ejecutar la acción de guardarlos necesita información sobre su ubicación predeterminada y cómo llegar a ella. Además, posiblemente realizando la orden necesite resolver otro tipo de actividades emergentes. Por ejemplo, cuando el refrigerador ya no tiene espacio disponible para guardar más cosas o cuando las rutinas de reconocimiento de objetos fallan en reconocer un artículo y debe tratar ese objeto con los valores por defecto de la categoría.

Además, es posible que el robot esté conectado a otro tipo de dispositivos de iluminación o temperatura. Esto puede llevar al robot a participar en interacciones como:

El usuario hace una consulta al estado del mundo.

- ¿Cuál es la temperatura de la sala de televisor?

El robot responde:

- El cuarto de televisor está a 30 grados.

El usuario da una orden que involucra conexión con software externo.

- Ajusta la temperatura del cuarto de televisión a 25 grados.

Esta situación ejemplifica que algunas órdenes que el robot puede procesar no siempre son ejecutadas por él, sino que se debe comunicar con otros sistemas que usan lenguajes formales para controlarse. Esto significa que las expresiones de significado formal de las oraciones deben ser flexibles para adaptarse a distintas sintaxis dependiendo de los sistemas computacionales a los que el robot tiene acceso.

Existe un gran número de distintas construcciones en el idioma inglés pero al menos cuatro son importantes y frecuentes en la interacción humano-robot: las estructuras de enunciados *declarativos* para manifestar hechos sobre el mundo que debe conocer

el robot, *imperativos* para que el robot ejecute un comando y *preguntas* las cuales se pueden dividir entre *de respuesta si/no* y las *preguntas con adverbios wh* (what, who, where).

La estructura declarativa relaciona dos conceptos ya conocidos o declara un concepto nuevo. Algunos ejemplos relevantes para un robot de servicio serían:

- *All bottles that contain water are cold*
- *The chips are on the table*
- *The exit is blocked*
- *Mary eats in the kitchen*
- *Table1 is an instance of table*
- *Thin is an adjetive of physical apperance*
- *Computer is a kind of electronic device*
- *Shape is a kind of attribute*

Con este tipo de declaraciones se agregan nuevos nodos que representen tanto categorías de objetos como categorías de atributos, así como relaciones entre ambos. Es decir, usando lenguaje natural se puede enriquecer la base de conocimiento.

Los enunciados imperativos comienzan con una frase verbal y normalmente no tienen una frase nominal que sea el actor de la acción (se infiere que el actor es el robot). Este tipo de oraciones son las que justifican el uso de un planeador automático para ejecutar el comportamiento solicitado, como en:

- *Bring me the newspaper from the office*
- *Stop*
- *Go to the livingroom*
- *Take the rubbish bin to the garage*

Estas oraciones no siempre expresan todos los roles semánticos importantes para cumplir la acción. Por ejemplo, *Take the rubbish bin to the garage* no especifica cuál es la ubicación actual del bote de basura por lo que el sistema debe de consultar información contextual para producir una interpretación completa.

Las oraciones que codifican preguntas de respuesta si/no empiezan con un verbo auxiliar o de ser/estar, seguido de al menos una frase nominal y opcionalmente finalizan

con una frase verbal. Estas estructuras parafrasean un hecho en el mundo que debe comprobarse en el contenido actual en la base del conocimiento, como en:

- *Is the beer cold?*
- *Is Mary eating in the kitchen?*
- *Are you ok?*
- *Does the fridge work?*
- *Is anybody home?*
- *Is there food on the table in the livingroom?*

Algunos enunciados que tienen esta estructura significan un comando o una sugerencia como *Can you bring the ketchup?*

Las preguntas con adverbios wh siempre contienen un componente llamado *wh-phrase* (WP) que a su vez contiene una *wh-word* (who, whose, when, where, what). Existen dos grandes clases dentro de esta clasificación, la clase de preguntas wh que tienen sujeto y las que no. Las preguntas que sí tienen sujeto siguen una estructura similar a los enunciados declarativos excepto que tienen una Wh-word al comienzo, algunos ejemplos son:

- *What is the color of the bottle of beer in the fridge?*
- *What is on the sofa?*
- *Who is in the livingroom?*
- *Where are the candies?*

Normalmente estas preguntas solicitan el valor de un atributo que califica a un objeto o clase, o consultan cuales clases y objetos cumplen restricciones expresadas en términos atributos y valores.

Otro tipo de interacciones a considerar son las que se presentan en la competencia robocup@home en la prueba *Enduring General Purpose Service Robot*. Esta prueba está diseñada específicamente para promover la comprensión de lenguaje natural y alejarse de especificar las pruebas como máquinas de estado. El robot recibe tres tipos de órdenes directas:

- Tres órdenes consecutivas cuyos parámetros son dados explícitamente como *Go to the living room, then take the coke and present yourself.*

- Una orden cuyos parámetros están subespecificados como *Bring a drink*. El robot puede pedir la información faltante o resolver la orden con la información de la oración y el modelo del mundo.
- Un comando como los anteriores pero que contiene información errónea sobre el mundo, por ejemplo *Bring a hot meal to the kitchen*, cuando en realidad no existe comida accesible por el robot. Nótese que esta categoría no difiere en lenguaje de las anteriores y su distinción debe reflejarse en la planeación automatizada y no en la interpretación de la oración.

2.2 Propuesta de solución

Al considerar ejemplos como los anteriores resaltan pautas típicas en las interacciones:

- Las frases nominales de las oraciones hacen referencia a objetos en la representación del mundo. Los cuales a su vez son abstracciones de objetos y lugares reales que sirven como parámetros para las rutinas de percepción.
- Los verbos en las oraciones que significan órdenes para el robot tienen una relación directa con el tipo de acciones que puede ejecutar. Por ejemplo, los verbos *bring*, *get* y *fetch* tienen normalmente el mismo sentido de recolectar objetos.
- Los parámetros de la actividad que debe resolver el robot no siempre están estipulados en el enunciado y deben obtenerse del contexto en que se da la interacción. Comúnmente algunos complementos circunstanciales se omiten porque están implícitos en el contexto de la interacción o en el historial de la conversación. Por ejemplo, en la orden *Bring a coke* se puede asumir que la coca está en el último lugar donde se supo de su ubicación y que el lugar destino para ese objeto es el mismo sitio donde está el robot al momento de escuchar la orden.

Tratar de enumerar cada una de las frases que el robot debe comprender es impráctico, por lo que es necesaria una forma compacta de especificar el significado que subyace a un conjunto grande de enunciados. La especificación de estos conjuntos de frases junto con su expresión formal de significado forman el lenguaje natural controlado que acepta el intérprete de lenguaje.

Por otro lado, la planeación de actividades en cualquier sistema requiere primero especificar las acciones que la máquina puede ejecutar de manera directa, qué tipo de actividades requieren un proceso de deliberación para crear una secuencia de dos o más acciones y cuáles son los escenarios que pueden obstruir el proceso de planeación de su curso natural.

De los escenarios descritos en la subsección anterior se resume que el planeador debe soportar mecanismos para:

- Expresar una sola acción de alto nivel con la sintaxis y parámetros adecuados para que otro programa en la arquitectura de software del robot coordine hardware en la máquina y realice cambios en el mundo real. Esto define lo que posteriormente se refiere como *tareas primitivas* e incluyen coger cosas, cambiar de posición, responder verbalmente, entre otras.

- Proponer una secuencia de acciones de alto nivel según algún paradigma de planeación automatizada que en su conjunto respondan a la orden solicitada.

- Identificar escenarios donde las acciones planeadas no correspondan con lo esperado y proponer acciones emergentes para provocar un estado del mundo válido para continuar con el plan original.

El primer caso es posible resolverse casi en su totalidad a nivel de interpretación de lenguaje, ya que toda la información necesaria está disponible en el estado del mundo inicial. No obstante, para el resto de los casos se necesita una forma de descomponer una orden de mayor grado de complejidad en una secuencia de acciones ordenadas. Existen distintos paradigmas de planeación automática que se pueden aplicar exitosamente a robots en contextos como este. Estos incluyen máquinas de estado, búsqueda en posibles configuraciones del micromundo, planeadores usando probadores de teoremas y usando lenguajes de programación para ambientes dinámicos [15] [16] [24].

Se plantearon las siguientes restricciones para el diseño del sistema:

- La base de conocimiento debe proveer mecanismos de inferencia mediante el uso de un subconjunto de la lógica formal.

- El proceso de interpretación del lenguaje debe hacer uso de información sintáctica y semántica del idioma, así como del conocimiento del mundo (contexto) almacenado hasta el momento y debe permitir especificar de forma sencilla el lenguaje formal que genera.

- El planeador de acciones debe arrojar órdenes de alto nivel que se consideran atómicas para el resto de la arquitectura dentro del robot, resolviendo situaciones contingentes que pueden o no suceder.

El propósito del sistema es la traducción de oraciones de lenguaje natural a una secuencia de acciones de alto nivel. Este proceso puede dividirse en dos pasos consecutivos:

1. **Asignación de significado del enunciado de entrada**. Formaliza el significado de la oración a una expresión que es parte de un lenguaje formal que usa el planeador de acciones.

2. **Generación de una secuencia de acciones de alto nivel.** A partir de una expresión de significado, el planeador usa el modelo del mundo y el modelo del robot para enumerar las acciones que resuelven la orden solicitada.

En este trabajo se asigna el significado de un grupo de oraciones de entrada como una expresión que pertenece al lenguaje formal que usa el planeador automático para iniciar el proceso de planeación. Estas expresiones tienen la forma

$(Tarea\_k : Rol_0 : Parametro_0, Rol_1 : parametro_1, \ldots, Rol_N : parametro_N)$

Esta expresión describe la actividad de alto nivel que el robot debe ejecutar y cada uno de los roles semánticos que corresponden al actor, objeto, lugar, tiempo de la acción o cualquier otro complemento que modifique la actividad. Para establecer un método de asociación de una oración en lenguaje natural a una representación de significado se necesitan algoritmos de distinta naturaleza que incluyen desambiguación léxica, inferencia lógica, análisis sintáctico superficial (*shallow parsing*), localización de roles semánticos, asociación de frases nominales a objetos del modelo del mundo, entre otros. Estos algoritmos usan información de lenguaje, del conocimiento sobre el mundo y la especificación del lenguaje de salida. En la figura 2.1 se muestra la información que la interpretación de oraciones requiere.

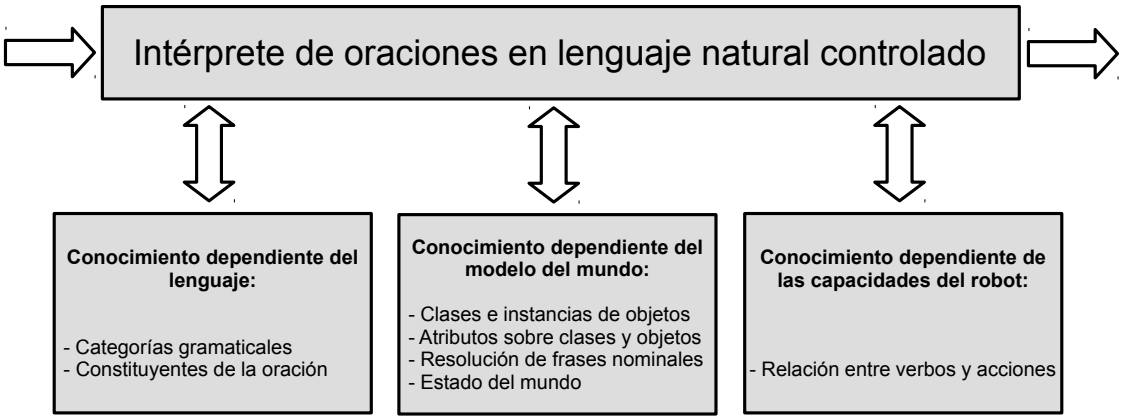


Figura 2.1 Tipos de conocimiento necesario para la interpretación de oraciones.

En la figura 2.2 se muestra una representación gráfica de las dependencias entre algoritmos que componen el intérprete de lenguaje es información requerida, donde los

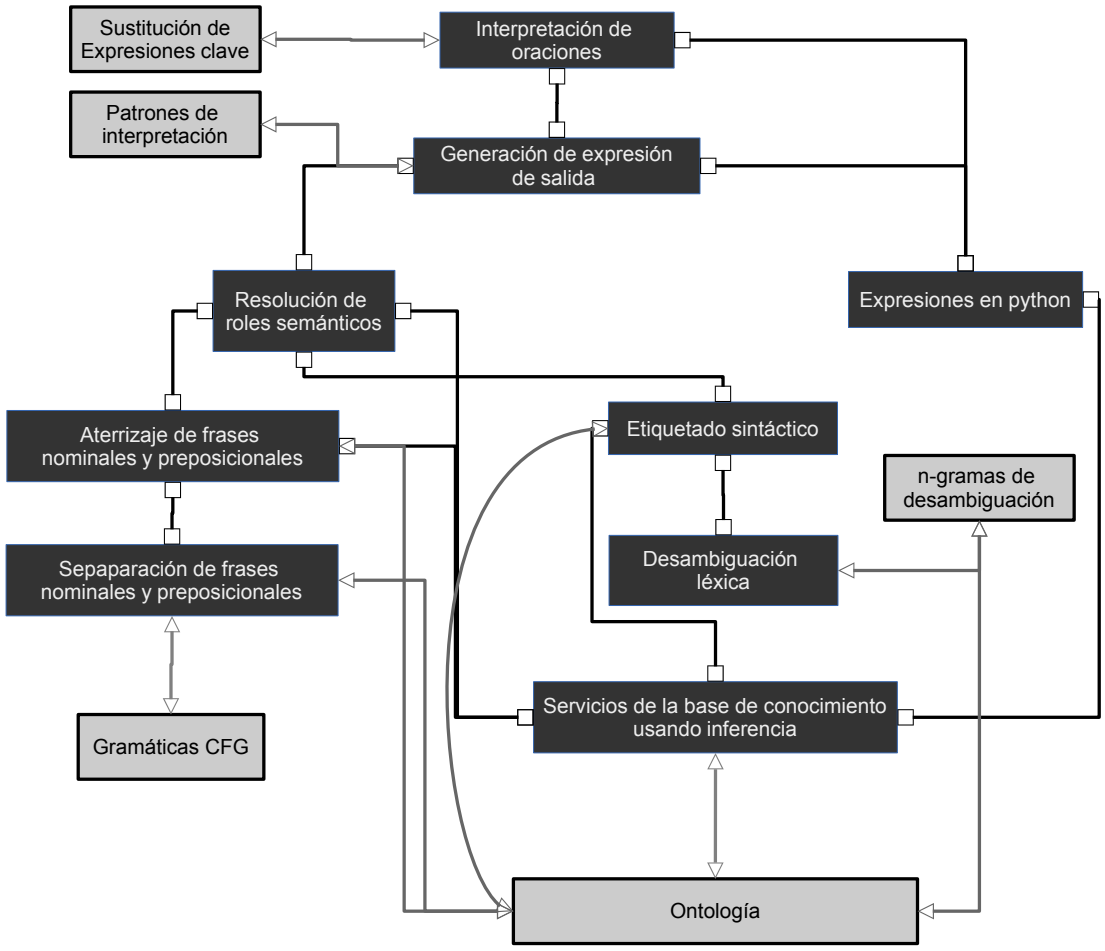


Figura 2.2 Dependencias entre los algoritmos que componen la interpretación de lenguaje.

cuadros oscuros son un proceso algorítmico y los cuadros claros son información que consultan los algoritmos. Así, el sistema usa los patrones de interpretación para generar las expresiones de significado. Procesar un patrón involucra información lingüística dependiente del lenguaje en forma de sustitución de expresiones clave, gramáticas y n-gramas. También integra conocimiento sobre el modelo y estado del mundo consultando la ontología a través de la base de conocimiento y tiene acceso a cualquier función externa a través del intérprete de desarrollo, en este caso *python*. El trabajo del intérprete de lenguaje es explorar la oración de entrada, el modelo del mundo y recursos externos para localizar y asociar un objeto en la ontología con los parámetros descritos, conformando una expresión final, tal que pueda procesarse por el planeador automático.

En el cuadro 2.1 se muestra cómo se conforma un patrón de interpretación. En este trabajo propongo estos patrones de interpretación como una forma compacta, expresiva y fácilmente computable para asignar un significado a distintos tipos de frases. Estos patrones describen una actividad que el robot debe resolver junto con los roles semánticos que se requieren para describirse totalmente. Cuando se logra asignar un objeto en la ontología a cada rol semántico entonces la expresión de salida queda totalmente definida y el enunciado se considera interpretado. La forma de describir cada rol se denota mediante:

- **Constituyente o categoría gramatical**. Es la etiqueta o estructura sintáctica del que proviene ese parámetro, como puede ser un sustantivo, una frase nominal o una frase preposicional.
- **Tipos semánticos compatibles**. Son categorías de objetos representados en el modelo del mundo, como son lugares, objetos, personas, entre otras.
- **Palabras clave**. Palabras como *from* o *to* que deben estar contenidas en constituyente las cuales ayudan a diferenciar entre parámetros similares.
- **Valor por defecto o fuentes externas**. Cuando la información sobre el parámetro está ausente en el enunciado, se debe consultar otra fuente de información, como la base de conocimiento o un agente de percepción.

Expresión de salida	(funcion_X(funcion_1(Rol_1 : Parametro_1), ..., funcion_N(Rol_N : Parametro_N)))			
Roles	**Constituyente**	**Tipos semánticos**	**Palabras clave**	**Consulta externa**
Parametro_1	consituyente_1	tipo_11, ..., tipo_1k	palabra_11, ..., palabra_1j	funcion_1(A, B, ...)
...	...	...	...	...
Parametro_N	consituyente_n	tipo_n1, ..., tipo_nk	palabra_n1, ..., palabra_nj	funcion_n(C, D, ...)

Cuadro 2.1 Patrón genérico de interpretación de oraciones.

Para asociar una oración de entrada en lenguaje natural a una expresión formal, primero se resuelven todos los roles semánticos o parámetros de la expresión de salida y después se evalúan las expresiones de consulta a medios de información externos para constituir la expresión final. La expresión final cumple con las cualidades de sintaxis y tipo de parámetros adecuados para procesarse en el planeador de tareas. Lo anterior implica que el lenguaje aceptado por el sistema está dado por el conjunto de estos patrones que definen cómo interpretar la oración de entrada. Si el enunciado no se puede interpretar mediante ninguno de los patrones, no se le puede asignar un significado formal completo y no pertenece al lenguaje natural controlado aceptado. Cada interacción

verbal que se quiera procesar puede tener muchos casos y consideraciones, sobre todo si se quieren interpretar oraciones con muchos parámetros, por lo que es posible que se necesiten múltiples patrones para abarcar todas las posibles paráfrasis que se desea aceptar. No obstante, por el grado de abstracción general o específica con el que se describen los roles semánticos, es relativamente fácil diseñar patrones de interpretación que cubran múltiples paráfrasis cada uno.

Para poder adaptar el sistema a otro tipo de micromundos es conveniente separar este tipo de información lingüística y léxica de los mecanismos descritos en código a nivel de implementación. Es decir, debe evitarse hacer suposiciones en el código sobre la información que se utiliza para dar un significado formal a una oración, así las decisiones particulares sobre el significado de la frase se hacen en el diseño del patrón de interpretación, gramáticas, n-gramas, etcétera.

Una vez completo el proceso de interpretación de significado, la descripción de la actividad se envía al planeador automático. Dicho planeador usa el paradigma de redes jerárquicas de tareas. Una actividad se describe como un grafo dirigido acíclico como en la figura 2.3 donde las aristas representan el orden de la ejecución de las tareas y los nodos representan una actividad que está descrita a su vez por otra red de tareas o una tarea primitiva que el robot puede ejecutar directamente. Cada sustitución o ejecución de la actividad que describen los nodos está sujeta a precondiciones y efectos sobre el modelo del micromundo.

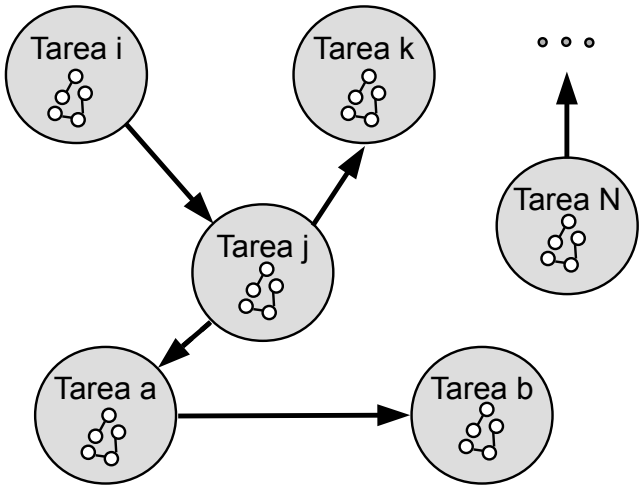


Figura 2.3 Especificación del espacio de planes como una red de tareas jerárquicas.

Cada nodo en el grafo representa una tarea que el robot puede ejecutar y está descrito mediante:

- **Parámetros de entrada**: Son objetos que se requieren especificar para ejecutar la tarea en el planeador de acciones y forman parte del conjunto de los objetos que fueron asociados a un rol semántico durante el proceso de interpretación de la oración.
- **Precondiciones**: Son un conjunto de hechos que describen el estado del mundo apropiado para poder ejecutar esa tarea.
- **Efectos**: Son un conjunto de hechos que se vuelven o dejan de ser verdaderos al terminar la tarea y corresponden a los cambios que el robot provoca al modificar su ambiente.
- **Comando a agentes externos o red de sustitución**. Cuando la tarea es primitiva incluye una expresión formal que se debe enviar a un módulo externo que cumple un propósito específico, como trasladar al robot, coger objetos, etc. Cuando la tarea no es primitiva se sustituye por la red de subtareas que describe la tarea de forma más específica, manteniendo las ligas de orden de ejecución.

En consecuencia, el proceso de interpretación de lenguaje y de planeación automatizada son dos procesos separados que comparten información sobre el modelo del mundo mediante la base de conocimiento.

Capítulo 3

Representación de conocimiento simbólico

La representación del conocimiento es el estudio del uso formal de símbolos para representar una colección de preposiciones que se consideran verdaderas o falsas. Estos símbolos no suelen representar explícitamente toda la información de la base del conocimiento. Es necesaria la *inferencia* sobre lo explícitamente dado para extraer conocimiento implícito. En este contexto, la inferencia es la manipulación de los símbolos en una base de conocimiento para producir nuevas proposiciones consistentes [20].

La representación del conocimiento es un aspecto importante a resolver en inteligencia artificial. Newell propone al conocimiento como una parte funcional dentro de un sistema inteligente que resuelve un tarea específica, el cual debe estar integrado con un sistema de percepción, sistema motor, sistema de procesamiento, etc. La representación del conocimiento es la estructura de datos que sostiene a la información de la especificación de la tarea y todos los datos necesarios sobre el mundo para que se interprete adecuadamente. El hecho de que una estructura de datos represente un concepto se debe a los procesos que manipulan la información de una manera consistente, lo cual implica más que rutinas para borrar o añadir datos. Se trata de todo un sistema activo que usa esas estructuras de datos. Sin embargo, para Newell el conocimiento en sí mismo es una noción diferente, que está en un nivel superior a la manipulación de estructuras de datos con sistemas lógicos y el cual tiene un rol en la naturaleza intrínseca de la inteligencia [22].

John McCarthy enuncia que *"La clave para alcanzar la inteligencia humana en inteligencia artificial es hacer sistemas que operen adecuadamente en situaciones informáticas de sentido común."*. En estas situaciones los hechos almacenados pueden estar incompletos y no existe certeza apriori de su relevancia. El sistema necesita definir conceptos de forma aproximada y valores por defecto para llegar a conclusiones. Además requiere conocimiento sobre el estado mental del agente [23].

Lemaignan enuncia que el conocimiento está compuesto por declaraciones que están contextualizadas, aterrizadas y limitadas a un dominio válido. Por *contextualizar* se refiere a ligar una declaración en la base de conocimiento con un contexto cultural, una referencia de interpretación adecuada para relacionar los hechos previamente adquiridos de forma válida. El *aterrizaje* se refiere a la identificación, creación y mantenimiento de un símbolo, como lo es el nombre de un objeto, con su verdadero significado en el mundo real. El *dominio de validez* especifica el alcance o rango de situaciones en las que un hecho debe considerarse verdadero. Esto puede incluir restricciones de duración temporal o probabilistas [24].

Estos aspectos llevan a formular los requisitos de la representación en dos perspectivas: los elementos que son necesarios para el conocimiento y los procesos o mecanismos necesarios para producir y extraer conocimiento. Los elementos necesarios para el conocimiento son:

- Conexiones, relaciones entre las unidades atómicas indivisibles del conocimiento.
- Conocimiento general sobre el mundo.
- Restricciones sobre la validez del conocimiento.

Además, la base de conocimiento debe contener los siguientes mecanismos:

- Recibir conocimiento del mundo a partir de sistemas de percepción o de otras fuentes, como humanos e incluso otros robots.
- Conectar nuevos hechos a hechos previamente almacenados.
- Monitorear hechos almacenados y manejar sus interconexiones y validez.
- Asegurar la consistencia del conocimiento almacenado.

El principio de que el conocimiento es mejor representado usando lógica formal ha sido debatido desde los inicios de la representación del conocimiento. Algunos creen que la lógica de primer orden es un lenguaje apropiado para capturar la esencia del razonamiento racional debido a su expresividad y su poder de inferencia. No obstante, otros creen que el conocimiento en el sentido más general tiene problemas fundamentales para enumerarse y representarse a través de un lenguaje formal [26].

3.1 Lenguajes de representación de conocimiento

En el contexto de esta tesis, el principal papel del lenguaje de representación es proveer un mecanismo de representación adecuado y formal para modelar conocimiento

con el fin de utilizarse en la interpretación de lenguaje natural y en la planeación de acciones de alto nivel. La mayoría de los lenguajes de representación parten de algún subconjunto de la lógica formal. La elección de tal subconjunto tiene impacto en el rango de hechos que se pueden expresar de manera fácil (expresividad práctica) y si acaso se pueden expresar (expresividad teórica) [24].

Brachman y Levesque describen que existe un equilibrio entre la expresividad de un lenguaje y la eficiencia del manejo de la inferencia. En general, un lenguaje con un grado de expresividad bajo puede usar algoritmos de inferencia eficientes computacionalmente. En contraste, un lenguaje expresivo permite almacenar hechos más complejos a costa de necesitar complejos algoritmos de inferencia y extracción de información [25].

3.1.1 Lógicas de descripción

En la lógica de primer orden es común representar categorías de objetos con un sustantivo usando predicados como *compañía(X), empleado(X), contrato(X)*. Sin embargo, en lenguaje natural no todas las frases nominales son simples sustantivos. Para capturar frases nominales más complejas es necesario que los predicados tengan una estructura interna. Por ejemplo si se quisiera representar la frase nominal *un hombre cuyos hijos son todos hombres* se podría construir el predicado compuesto $padre\&tiene\_solo\_hijas(X)$

Para considerar este tipo de descripciones manipulando predicados compuestos se ha desarrollado el formalismo de representación de conocimiento conocido como *lógicas descriptivas o lógicas de descripción* (DL) [20]. Este formalismo representa el conocimiento sobre el dominio de una aplicación al definir conceptos relevantes (terminología) y al usar estos conceptos para especificar propiedades de objetos e individuos (estado del mundo). En DL, los conceptos se organizan jerárquicamente en una ontología. Sin embargo, no todas las descripciones de conceptos se estipulan explícitamente en la ontología sino que se obtienen al manipular múltiples *conceptos y roles atómicos* mediante *constructores de conceptos* [27].

Las lógicas descriptivas han sido usadas en una gran variedad de aplicaciones incluyendo la integración de datos y la respuesta automatizada de preguntas. También es la base teórica del *lenguaje de ontologías web* o *web ontology language* OWL [1] [26].

[1] El lenguaje OWL se desarrolló por el grupo de ontologías web W3C bajo la premisa de diseño de proveer la expresividad de las lógicas descriptivas para formalizar el conocimiento de la web. OWL tiene dos variantes, *lite* y *DL completo*. Ambas variantes pueden explicarse como un Tbox junto a una jerarquía de roles, describiendo el dominio en términos de clases y propiedades. Las decisiones de diseño durante la creación de OWL permiten que se exploten muchos resultados teóricos sobre la complejidad y decidibilidad de las DLs, así como el uso de algoritmos de razonamiento desarrollados con anterioridad y ha permito la rápida asimilación de esta tecnología en la *web semántica* [26].

3.1.2 Sistemas de reglas de producción

Otro paradigma de representación de conocimiento es el basado en reglas las cuales dado un antecedente en la representación del mundo infieren un consecuente. Una regla puede entenderse de manera procedural como:

- Partiendo de las aseveraciones P, se asevera que Q *(aseverar P)* $\rightarrow$ *(aseverar Q)*, o
- Partiendo de las metas Q, se tienen las metas P *(meta Q)* $\rightarrow$ *(meta P)*

Aunque ambas opciones se concluyen de la relación entre P y Q, se enfatiza la diferencia entre aseverar hechos y buscar el cumplimiento de metas. En general, estos dos tipos de razonamiento se nombran como *razonamiento dirigido a datos* y *razonamiento dirigido a metas*. En este trabajo uso el primer tipo de razonamiento para especificar algunos servicios de la base del conocimiento y el segundo como método de implementación para el planeador de acciones.

Un sistema de producción es un sistema de razonamiento *encadenamiento hacia adelante* que emplea reglas llamadas *reglas de producción* que representan conocimiento general. Un sistema de producción mantiene los hechos almacenados en una memoria de trabajo (MT). La MT constantemente cambia durante la operación del sistema.

Por otro lado, una regla de producción es una estructura de dos partes: el antecedente como un conjunto de precondiciones y el consecuente como un conjunto de efectos. Las precondiciones son pruebas que se aplican al estado actual de la MT para determinar si la regla puede aplicarse. El consecuente son los hechos que se vuelven verdaderos cuando una regla se aplica y por tanto modifica la MT. Entonces la operación básica de un sistema de producción es un ciclo de tres pasos que se repiten hasta que ninguna regla es aplicable en el estado de la MT. En este punto el sistema se detiene [2]. Los tres pasos del ciclo son [20]:

1. **Reconocer.** Encontrar cuáles son las reglas cuyas precondiciones se satisfacen con el estado actual de la MT.
2. **Resolver conflictos.** Elegir una regla a ejecutar dentro de todas las reglas encontradas en el paso 1 (no se pueden ejecutar múltiples reglas simultáneamente).
3. **Actuar.** Actualizar la MT acorde al conjunto de acciones en el consecuente de la regla que se disparó.

[2]Un lenguaje de programación que sigue este paradigma de cómputo es *C Language Integrated Production System* CLIPS, desarrollado por la NASA en 1985 pero actualmente es software de dominio público mantenido por programadores sin relación con la agencia espacial estadounidense [28].

3.1.3 Ontologías

Aunque las ontologías actualmente son conocidas como artefactos computacionales, en realidad se originaron como una disciplina de la filosofía dedicada al estudio de la existencia y la descripción formal de la realidad. De acuerdo con Barry Smith *"La ontología como rama de la filosofía es la ciencia de lo que es, los tipos y estructuras de los objetos, propiedades, eventos, procesos y relaciones en cada aspecto de la realidad"* [36].

En computación, de acuerdo con Guarino *"Una ontología es conjunto de axiomas que expresan la intención de significado de un vocabulario formal. Un modelo de lenguaje que use dicho vocabulario comparte el compromiso de intención de la ontología ..."* [38]. Las restricciones del modelo entonces se expresan como relaciones entre conceptos en el espacio del dominio a representar. Una ontología es independiente del lenguaje natural pero está comprometido a la conceptualización del mundo por medio de un lenguaje de representación. Una ontología provee un conjunto de definiciones los cuales restringen el significado del vocabulario. Las ontologías también pueden codificar hechos sobre el mundo usando relaciones entre individuos o clases [38]. Estas relaciones se pueden dividir en [36]:

- **Relaciones de subclase.** Definen subcategorías entre conceptos. Por ejemplo, una iglesia es una especialización de edificio en el sentido que cualquier subclase o individuo de iglesia está comprometido a ser a su vez un edificio.
- **Relaciones no taxonómicas.** Relacionan dos conceptos mediante una relación que codifica un hecho sobre el mundo. Por ejemplo, la relación *has_style* puede relacionar a una iglesia con su respectivo estilo arquitectónico.
- **Restricciones sobre dominio o rango sobre relaciones.** Definen qué tipo de conceptos son objeto de una relación determinada. Por ejemplo, la relación *has_style* requiere unir un objeto del tipo *Arquitectonic_artefact* con un *architectonic_style*.
- **Restricciones de cardinalidad.** Definen restricciones sobre la validez de una categoría al cumplirse la cardinalidad de otra relación. Por ejemplo, una catedral necesita al menos tener una cúpula.
- **Relaciones de parte.** Establecen que un objeto es un elemento dentro de un conjunto que conforma el concepto más amplio. Por ejemplo, el campanario es parte de una catedral.
- **Disyunciones.** Establecen separaciones entre clases o relaciones. Por ejemplo, un objeto de clase catedral no puede ser simultáneamente un objeto de clase estilo arquitectónico.

3.2 Diseño de la base de conocimiento para un robot de servicio doméstico

El principal reto de la representación del conocimiento puede plantearse como *la forma de modelar de manera simbólica estados e interacciones del micromundo de una casa habitación para que un robot doméstico pueda tomar decisiones de sentido común.*

La correcta elección de una estructura de almacenamiento puede facilitar el diseño de las rutinas de consulta e inferencia. Tomando en cuenta los trabajos en [2], [11], [15], [24] y [31] decidí usar una ontología de conceptos sobre la cual se construyen los mecanismos de inferencia para establecer implicaciones sobre el conocimiento explícito. Además, los conceptos de la ontología sirven simultáneamente como un léxico para el procesamiento del lenguaje natural controlado.

La forma en que se diseña el contenido de la ontología sigue una política *de especificación*, por lo que cada concepto es un refinamiento de otra categoría más general. Los nodos terminales en el grafo que conforma la ontología corresponden a instancias reales de las categorías de objetos.

Debido a que la ontología puede almacenar cualquier símbolo arbitrario como un concepto u objeto, la base de conocimiento podría usarse para almacenar cualquier tipo de información que se pueda representar por medio de una secuencia de caracteres. Por ejemplo, las rutas de las imágenes de entrenamiento para las rutinas de reconocimiento de imágenes de un objeto y las coordenadas geométricas de la ubicación de una persona. Sin embargo, debido al alcance de la tesis decidí diseñar la base de conocimiento a un nivel de granularidad mayor, donde se enfatiza su uso tanto en procesamiento de lenguaje como en planeación automatizada.

3.2.1 Diseño de la ontología

Para integrar un diseño reconfigurable para otros micromundos similares, conviene mantener pautas de diseño que no asuman restricciones severas sobre el modelo del mundo.

Debido a la estructura de datos inherente a una ontología, la unidad fundamental de información es una terna de símbolos del tipo *concepto1, relación, concepto2* lo cual es equivalente en notación de predicados lógicos binarios como *relación(concepto1, concepto2)*. Las relaciones entre conceptos se llaman en este trabajo *atributos* para evitar confusión con los roles semánticos de una oración y corresponden a la etiqueta del vértice que une dos nodos. Cada atributo está declarado igualmente en la ontología como una jerarquía de atributos. Un nodo en la ontología puede representar alguna de las siguientes entidades:

- **Una categoría de conceptos u objetos.** Forma la jerarquía de objetos dentro del modelo del mundo. Por ejemplo, las categorías *lugar, mueble* y *persona*.

- **Una categoría de atributos.** Forma la jerarquía de propiedades de los objetos y se presentan directamente en lenguaje como clases de adjetivos. Por ejemplo, las categorías *estado civil, forma* y *color*.

- **Una instancia de una categoría de objetos.** Representa una abstracción de un elemento real del mundo, el cual puede ser manipulado o percibido por el robot. Por convención al módulo de interpretación de lenguaje, estos símbolos corresponden con los nombre propios de cada objeto. Por ejemplo, *mary, mesa_1, mesa_cocina*.

- **Una instancia de un atributo.** Representa una propiedad que califica a un objeto o categoría y se considera un adjetivo calificativo en lenguaje natural. Por ejemplo, *soltero, redondo, azul*.

Decisiones sobre el contexto cultural del dominio del micromundo se hacen en la ontología, por lo que el desarrollador de la aplicación debe poder modificar el contenido de la jerarquía de conceptos y atributos para lograr que la interpretación de oraciones cumpla con los requisitos deseados. Sin embargo, para formar los servicios de inferencia sobre los datos de la ontología se tuvieron que definir tres nodos especiales obligatorios los cuales se consideran a nivel de algoritmo. Estos nodos especiales sirven para diferenciar las dos grandes categorías de conceptos consideradas: objetos y atributos.

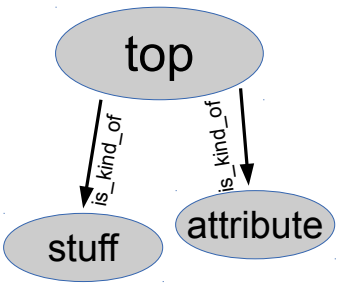


Figura 3.1 Ontología incial.

Como se ve en la figura 3.1, existe una categoría general llamada *top* que se subdivide en *stuff y attribute*. A partir del nodo *stuff* se refinan categorías e instancias de objetos, lugares, personas, etc. De *attribute* se refinan categorías de las relaciones que se aplican sobre un par de subcategorías de *stuff* como *ubicación, forma, temperatura*, entre otras.

Asimismo, existen cinco relaciones especiales que tienen un propósito específico dentro de los algoritmos de consulta e inferencia. Estos atributos son: *is_kind_of, is_object_of, attribute_property y aka.*

- **is_kind_of**. Sirve para declarar que una clase es un refinamiento de otra. Por ejemplo en *is_kind_of(perro, mamífero).*
- **is_object_of**. Sirve para declarar que un nombre propio hace referencia a una instancia real de una categoría. Por ejemplo en *is_object_of(don_gato, gato).*
- **aka**. Sirve para asignar un alias a cualquier nodo. Es la abreviatura de *Also Known As.* Por ejemplo, *aka(perico, loro).*
- **attribute_property**. Sirve para declarar las propiedades algebraicas cumple un atributo. Estas propiedades son: *transitividad, simetría, heredabilidad, y cardinalidad.*

3.2.2 Servicios de consulta

A continuación se describen algunas funciones importantes del módulo de la base de conocimiento, mencionando su propósito y utilidad.

- **Consulta de categorías padre e hija**. Recorre el grafo primero por anchura (se permite *herencia múltiple*) recolectando las clases que unen de manera transitiva a un nodo con todos sus categorías relacionadas mediante la relación *is_kind_of.* Por ejemplo, las clases *hombre* y *mujer* son *hijas* de la clase *persona.*
- **Consulta de los objetos de una clase**. Recorre el grafo primero por anchura (se permite *herencia múltiple*) recolectando las instancias declaradas sobre un categoría o cualquiera de sus refinamientos. Por ejemplo, si se consultan los objetos de la clase *persona* también se recolectarían las instancias de las clases *hombre* y *mujer.*
- **Consulta de los valores de una categoría de atributos en una clase u objeto**. Esta función recibe un nodo que representa una categoría de atributos y un nodo que representa una categoría de objetos. La función *get_values(my_object, some_attribute)* regresa las relaciones del tipo *attribute_k(my_object, value_i)* donde *attribute_k* es algún refinamiento de *some_attribute* y *value_i* es el valor del atributo *attribute_k* asociado al objeto *my_object.* Es importante enfatizar que la existencia explícita del hecho *attribute_k(my_object, value_i)* no está garantizada y para resolver esa consulta es necesario usar las propiedades algebraicas de cada *attribute_k* para extraer información implícita. Por ejemplo, se tiene la categoría de atributos *location* cuyos refinamientos son *adyacent y contained*, la categoría

adyacent tiene como objetos a *near, far, in_front* y la categoría contained tiene como objetos a *in, inside, on, at*, si se solicita la consulta *get_values(my_object, location)* la función regresa la representación de un grafo con vértices del tipo *in(my_object, fridge), inside(my_object, kitchen), near(my_object, kitchen_table)* y toda la información similar que relacione a *my_object* con cualquier refinamiento de *location*.

De forma similar al trabajo hecho en el robot *Golem II+* presentado en [30], la base de conocimiento permite la herencia de propiedades de una clase hacia todos sus refinamientos. De esta forma, un objeto puede exhibir una propiedad declarada de forma *particular* (declarada sobre ese objeto) o declarada de forma *general* (declarada sobre una clase padre). Para resolver conflictos entre propiedades particulares y generales se sigue el *criterio de especificidad*, el cual antepone propiedades particulares sobre las generales.

Capítulo 4

Formalización del lenguaje

En ciencias de la computación, trabajar con lenguaje natural tiende a establecer una secuencia de procesos por los que se debe llevar el texto crudo. Dicha secuencia concuerda con la separación entre análisis sintáctico, análisis semántico y análisis pragmático. El análisis sintáctico encuentra la estructura de la oración de tal forma que es posible asociar significados semánticos a las palabras. El análisis pragmático analiza el propósito del discurso y no solo de una frase aislada; es decir, analiza la oración en el contexto de oraciones previas [36].

En inteligencia artificial, el *entendimiento de lenguaje natural o natural language understanding, NLU,* es una rama de procesamiento de lenguaje natural. El objetivo de NLU es la correcta interpretación de una entrada de texto. Este proceso se puede conceptualizar como la traducción de un texto en lenguaje natural a una representación no ambigua en un lenguaje formal [38].

Debido a la aplicación que se le da a este trabajo, se discutirá aspectos del lenguaje haciendo énfasis en el idioma inglés, no obstante los ejemplos e ideas descritas pueden adaptarse para otros lenguajes naturales como el español.

4.1 Fundamentos sintácticos

La estructura sintáctica de los enunciados indica la forma en que las palabras se relacionan unas con otras, cómo se agrupan para formar frases y qué palabras modifican a otras. Algunas representaciones sintácticas se basan en la noción gramáticas libres de contexto, la cual permite describir la estructura de los enunciados en función de frases embebidas en frases de mayor complejidad [35].

4.1.1 Categorías gramaticales

Una categoría gramatical o *part of speech* POS es la categoría lingüística de las palabras que generalmente está definida por el comportamiento sintáctico o morfológico de la palabra. Algunas de las más comunes son *sustantivo, verbo, adjetivo y adverbio*. Cada lenguaje puede tener un número distinto de categorías gramaticales. Más aún, analizando un lenguaje en específico distintas convenciones proponen distintas categorías. Por ejemplo, para el idioma inglés la convención *Penn Treebank* que tiene 45 categorías, la del Brown Corpus tiene 85 y la C7 tagset tiene 146 [37].

Existen dos tipos de categorías gramaticales: las categorías *abiertas* y las *cerradas*. Las categorías gramaticales cerradas son un conjunto finito de palabras bien especificadas y muy rara vez se agregan nuevas palabras en esas categorías. Ejemplos de categorías cerradas son las preposiciones y los artículos. Por el contrario, las categorías abiertas cambian todo el tiempo porque algunas palabras entran en obsolescencia y otras nuevas son acuñadas con su uso. Ejemplos de categorías abiertas son los sustantivos, verbos, adjetivos y adverbios [35], [37].

Al proceso de asignar una categoría gramatical a cada palabra dentro de un texto se le conoce como *etiquetado de categoría gramatical*. Este etiquetado difiere al proceso de *análisis léxico* en compiladores de lenguajes computacionales por la naturaleza ambigua del lenguaje natural [37]. A pesar de que se tenga una tabla con los pares de *palabra y categoría gramatical* existen palabras que pueden funcionar como distintas categorías dependiendo de palabras previas o posteriores. Es decir, existe ambigüedad respecto a qué categoría asignar. En el enunciado *The cleaner is broken*, la palabra *cleaner* es un sustantivo. En cambio en *The cleaner, the better* la misma palabra es un adjetivo comparativo.

Existen distintos algoritmos diseñados para resolver la ambigüedad léxica y la mayoría de ellos caen en una de dos categorías: los algoritmos basados en reglas y los estocásticos. Los basados en reglas utilizan múltiples reglas de desambiguación hechas a mano. Estas reglas son del tipo *una palabra ambigua es un sustantivo en lugar de un verbo si es la palabra siguiente a un determinante*. Por otro lado, los etiquetadores estocásticos usan corpus etiquetados a mano como una fuente de ejemplos para el entrenamiento de algoritmos para poder computar la probabilidad de la categoría gramatical dado el contexto de la palabra, algunos de estos algoritmos usan cadenas de Markov para dicho cálculo [37].

4.1.2 Constituyentes de una oración

Cuando las palabras se asocien a su categoría gramatical correspondiente, es posible encontrar estructuras sintácticas en la oración que se comporten como una sola unidad o frase llamada constituyente. Por ejemplo, algunos constituyentes del inglés son:

- **Frase nominal (Noun Phrase, NP)**: Incluye nombres propios como *John*, pronombres como *she* o frases como *The intelligent robot*.
- **Frase preposicional (Prepositional Phrase, PP)**: Como *from the kitchen*.
- **Frase verbal (Verb Phrase, VP)**: Como *arrived on time*

Frecuentemente el orden en que se presentan los constituyentes puede variar sin alterar el significado de la oración y se dice que tienen una construcción *prepuesta* o *pospuesta* dependiendo el caso. Por ejemplo, la frase preposicional *Before I get home* tiene distintas posiciones en los siguientes ejemplos sin que eso afecte el significado de la oración:

- Before I get home I'd like you to turn on the air conditioner
- I'd like you to turn on the air conditioner before I get home

Para saber si un grupo de palabras forman un constituyente es común analizar la estructura sintáctica que presentan mediante gramáticas libres de contexto. Usando las reglas de producción de las gramáticas libres de contexto se pueden capturar la forma sintáctica de los constituyentes antes mencionados. Por ejemplo, el siguiente conjunto de reglas expresa que una NP (*noun phrase*) puede ser *pronombre* ProNoun, un *sustantivo propio* ProperNoun o un Det (Determinante como *the, a, an, those, any, some*) seguido de un constituyente Nominal, el cual a su vez puede ser varios sustantivos consecutivos [37].

$$NP \rightarrow ProNoun$$

$$NP \rightarrow ProperNoun$$

$$NP \rightarrow Det \quad Nominal$$

$$Nominal \rightarrow Noun \quad | \quad Nominal \quad Noun$$

Esta gramática es limitada y es fácil pensar en casos más complejos de requieren agregar más reglas de producción. El rol del determinante también se puede cumplir usando estructuras más complejas de palabras. Por ejemplo, en *John's friend's bike is great*, la función del determinante la realiza una expresión posesiva que consiste de una serie de NP seguidas del marcador de posesión *'s*. Las reglas de producción serían $Det \rightarrow Possessive$ y $Possessive \rightarrow NP's|NP'sPossessive$. También hay casos donde el determinante es opcional, como cuando el sustantivo que modifican es plural, como en *Bring me drinks that are sweet* o si el sustantivo no requiere determinarse (llamados *mass nouns*) como en *Breakfast is ready*.

El constituyente *Nominal* más simple es un sustantivo $Nominal \rightarrow Noun$, pero existen construcciones más complejas. Todo Nominal tiene una cabeza que es el sustantivo principal de la construcción. Antes o después de la cabeza pueden aparecer otras palabras que aporten información al nominal. Por su función sintáctica, a estas palabras se les conoce como *postdeterminantes* e incluyen clases de palabras como números cardinales, números ordinales y cuantificadores. Un ejemplo es *the first drink you see*, donde *first* es un postdeterminante. También pueden aparecer adjetivos entre un postdeterminante y el sustantivo como en *the last red object you brought*, donde *red* es un adjetivo. A su vez varios adjetivos pueden agruparse para formar una *frase adjetival*. Las palabras que van después de la cabeza se llaman postmodificadores (postmodifiers) y existen tres postmodificadores nominales comunes

- Frases Preposicionales o PP, como en All objects *on the table*
- *Non-finite clauses* o NFC, pueden hacer uso de verbos en gerundio, en terminación *ed* o en infinitivo, como en All people *asking for food*
- *Relative clauses* o RC que contienen un pronombre relativo (relative pronoun como *that, which, who*), como en The person *that is in the livingroom*

Además, existen palabras que aparecen antes de las NP que se llaman predeterminantes. Muchos de ellos se usan para contabilizar a la NP que preceden por ejemplo *All the empty bottles should be removed*, *All* es un predeterminante de la NP *the empty bottles*.

Por otro lado, las *Frases verbales* VP son constituyentes que consisten en un verbo rodeado de otros constituyentes, de lo antes mencionado se pueden proponer algunas reglas sencillas para definirlos.

$$VP \rightarrow Verb$$
$$VP \rightarrow Verb \;\; NP$$
$$VP \rightarrow Verb \;\; NP \;\; PP$$
$$VP \rightarrow Verb \;\; PP$$

Las VP pueden ser más complicadas. Después del verbo pueden aparecer un sinnúmero de constituyentes incluyendo otra VP como en *I want to drink beer* o incluso un enunciado entero como en *You said you were hungry*, por lo que se deben considerar también las reglas

$$VP \rightarrow Verb \;\; VP$$
$$VP \rightarrow Verb \;\; S$$

Una VP puede presentar una gran variedad de constituyentes que la modifican. Sin embargo, no todos los verbos son compatibles con todas las posibles estructuras de VP. Los verbos se suelen dividir por lo menos en: transitivos, como *find*, intransitivos, como *dissapear* y auxiliares que a su vez comprenden a los verbos modales como *can, may, must, will, shall* y a los auxiliares del tiempo perfecto, progresivo y pasivo, *have* y *be*, respectivamente. Algunas gramáticas modernas llegan a subcategorizar los verbos hasta en 100 clases [37].

4.2 Fundamentos semánticos

4.2.1 Recursos léxicos semánticos y ontologías

Los recursos léxicos semánticos contienen información sobre la organización semántica del léxico de un lenguaje en particular. Su objetivo es capturar algunos patrones o usos regulares de las palabras. En específico, las relaciones léxico-semánticas son un elemento central en la organización de las palabras. Existen muchas formas de subdividir esas relaciones una de ellas es en *relaciones paradigmáticas* y *relaciones sintagmáticas*. Las palabras que sostienen una relación paradigmática tienen la misma categoría gramatical, comparten algunas características e incluso se pueden sustituir una por otra en algunos contextos, por ejemplo las palabras *dog* y *animal*. Algunas relaciones paradigmáticas son sinónimos, hiperónimos y merónimos. En contraste, las relaciones sintagmáticas son relaciones entre palabras que ocurren una cerca de la otra en algún texto como *dog* y *bark*. Estas relaciones permiten por ejemplo agrupar las palabras en frases idiomáticas cuyo significado sólo puede entenderse al considerar la frase completa [38].

Comúnmente el lenguaje usa información sobre el mundo que debe hacerse explícita para poder extraer el significado completo de la frase. Los humanos hemos adquirido esa información a través de la educación académica, leer, reflexionar, conversar, etc. Ese conocimiento explica cómo está estructurado el mundo que conocemos. Hacer ese conocimiento explícito de tal forma que una computadora pueda consultarlo es un reto grande. Sin embargo, las ontologías pretenden hacer esta tarea de manera ordenada al coleccionar conocimiento explícito para usarse en la comprensión o solución de un problema donde esté involucrado conocimiento de sentido común. En la mayoría de casos, las ontologías tienen una extensión modesta y solo intentan capturar conocimiento de una tarea, situación o fenómeno específico.

Usando una ontologia es posible establecer relaciones léxicas como sinónimos, antónimos, hipónimos (subclases), hiperónimos (superclases), merónimos (es parte de), holónimos (tiene como parte a), etc. Sin embargo, estas relaciones no están dadas en nivel lógico sino lingüístico y se debe ser cuidadoso al mezclar un léxico con una ontología que

codifica conocimiento general. Además, una ontología normalmente almacena el lema de una palabra, es decir su forma básica. Frecuentemente se requiere mapear las palabras de lenguaje natural al lema almacenado en la ontología mediante un mecanismo adicional. Existen, intentos de mezclar un léxico con una ontología desde su diseño. Un ejemplo de la mezcla de información semántica léxica y una ontología es *ConceptNet* en la cual los nodos son fragmentos en lenguaje natural, los cuales están semiestructurados dependiendo de su patrón sintáctico [38].

4.2.2 Estructura de verbos

Las *gramáticas de casos* son gramáticas que enfatizan la relación de cada constituyente de la oración con el verbo o adjetivo principal. Usando estas gramáticas distintos fragmentos del enunciado se relacionan con distintos casos, también llamados *roles semánticos*. El conjunto de roles semánticos no está completamente definido para todos los verbos ni en todos los contextos ya que varía con el dominio y el nivel de abstracción del análisis que se requiera hacer. Por ejemplo, en el enunciado *John broke the window with a hammer*, el verbo *broke* une a las tres frases nominales *John*, *the window* y *a hammer*, *John* es el actor de la acción, *the window* es el objeto directo y *a hammer* es un instrumento usado en el evento *breaking the window*. En [36] se propone un conjunto de casos que incluyen a AGENT, INSTRUMENT, EXPERIENCER, BENEFICIARY, AT-LOC, AT-TIME, TO, FROM, entre otros. Dependiendo del verbo, algunos casos semánticos son obligatorios y otros no. Por ejemplo, el verbo *put* requiere de un actor, un objeto directo y una localidad como en *He puts the pencil down*. Sin embargo, el verbo *think* solo está obligado a tener un actor para tener un significado concreto como *She thinks* y otros roles semánticos son opcionales. Existen roles que son opcionales para todos verbos. Por ejemplo, todas las oraciones con verbos en pasado o futuro pueden tener un rol semántico para expresar el tiempo cuando se realizó o se realizará la acción.

Cada rol recibe únicamente una frase nominal y cuando varias frases nominales son necesarias por rol entonces deben estar unidas mediante una conjunción. Por ejemplo, en *John and I ran to the store* el rol semántico del actor está compuesto por *John and I* pero el enunciado *John I ran to the store* se consideraría mal formado [38].

4.2.3 Dependencias conceptuales

La teoría de dependencias conceptuales es una representación del significado de una idea desarrollada por Roger Schank presentada por primera vez en su tesis doctoral *A Conceptual Dependency Representation for a Computer-Oriented Semantics*. Esta teoría establece el significado de una oración como un grafo de dependencias entre objetos y roles semánticos [32]. Como existen muchas formas de expresar o parafrasear la misma

idea, se cree poco factible que los humanos guarden memoria de estructuras altamente relacionadas con el lenguaje natural. Schank considera más probable que se desarrolle un tipo de forma estándar del conocimiento, donde todas las posibles paráfrasis de una declaración se mapean a una forma canónica de significado.

Esta forma canónica de representación debe alejarse del lenguaje natural al evitar el uso de palabras o estructuras sintácticas ambiguas. Debe también resolver el uso de sinónimos, entre otros problemas.

La teoría de dependencias conceptuales tiene como premisa que una acción es la base de cualquier proposición que no es una descripción de un aspecto estático del mundo. Todas las proposiciones que describen eventos están hechas de conceptualizaciones, las cuales están formadas por una acción, un actor y un conjunto de roles que dependen de la acción. Se define una acción como algo que un actor puede aplicar sobre un objeto. Un actor es un objeto animado y un objeto es una entidad concreta en el mundo.

Para manejar las similitudes, diferencias y traslapes en el significado de las oraciones, Schank propone un conjunto finito de *acciones primitivas* que son las unidades básicas de significado con las cuales se puede construir una idea compleja. Estas acciones primitivas no son categorías de enunciados sino elementos independientes que se pueden usar en combinación unos con otros para expresar la idea que subyace una declaración. La teoría original se consideran doce actos primitivos siendo los más importantes:

- **ATRANS**: La transferencia de una relación abstracta. Por ejemplo la posesión, pertenencia o control. Requiere un actor, un objeto y un recipiente. Con este acto primitivo se pueden codificar verbos como *give, take, buy.*
- **PTRANS**: La transferencia de la ubicación física de un objeto. Requiere un actor, un objeto y una dirección o destino. Codifica verbos como *fly, go, walk, drive.*
- **PROPEL**: El uso de una fuerza física sobre un objeto. Requiere un actor, un objeto y una dirección. Codifica verbos como *push, pull, kick, crash.*
- **MTRANS**: La transferencia de una idea dentro o entre entidades animadas. Codifica verbos como *remember, see, tell, read.*
- **MBUILD**: La construcción de nuevo conocimiento en un animal. Codifica verbos como *remember, see, tell, read.*
- **INGEST**: Introducir un objeto dentro de otro.
- **GRASP**: Tomar un objecto.
- **ATTEND**: Concentrarse en percibir un estímulo sensorial.
- **SPEAK**: Articular palabras y sonidos.

- **MOVE**: Movimiento de una parte del cuerpo.

- **EXPEL**: Expulsar un objeto fuera del cuerpo.

El mayor atractivo de esta teoría es que se diseñó para representar la forma canónica del enunciado. Los roles conceptuales de cada caso no corresponden necesariamente a roles sintácticos y las palabras se procesan con base en sus relaciones conceptuales en el modelo del mundo. Los actos primitivos antes mencionados sirven para organizar el proceso de inferencia del significado de la oración. La idea de representar el significado usando acciones primitivas puede parecer razonable y práctica. Sin embargo, existe y existirá desacuerdo sobre la elección del conjunto de actos primitivos porque es difícil demostrar que absolutamente todas las ideas usando todas sus paráfrasis en lenguaje natural se pueden expresar usando un conjunto finito de actos primitivos [33].

4.3 Particularidades del lenguaje natural

4.3.1 Ambigüedad

Un vistazo rápido a cualquier diccionario muestra que las palabras tienen distintas interpretaciones o sentidos. Por ejemplo, la palabra *cook* tiene significado como el verbo *cocinar* y como el sustantivo *cocinero*. La palabra *still* puede ser el sustantivo *quietud*, el verbo *mantenerse quieto*, el adjetivo *quieto* o el adverbio *todavía*. A pesar de esta ambigüedad, los humanos casi no lo notamos en la vida diaria y descartamos los significados inapropiados automáticamente sin darnos cuenta. No obstante, un programa de computadora está destinado a racionalizar cada una de las opciones y descartar las interpretaciones equivocadas mediante algún mecanismo definido.

La ambigüedad además ocurre a distintos niveles de análisis, no únicamente al nivel léxico de las palabras. A pesar de que las palabras se relacionen adecuadamente con su significado, una frase puede interpretarse de múltiples formas. Por ejemplo, *Jack cooked Sue's dinner* puede significar que Jack cocinó la cena para Sue o que Jack cocinó la cena que pertenecía a Sue, no importando si era la intención original de Sue. Otras veces el enunciado puede tener una sola interpretación pero en realidad conlleva otra intención implícita. Por ejemplo, si alguien está en un concierto multitudinario rodeado de personas y la persona detrás dice *Excuse me, you are standing on my foot*, sería inapropiado comprender ese hecho como una declaración ya que conlleva una petición u orden de moverse para evitar pisar el pie del vecino. Sin tener conocimiento del contexto y del sentido común en cada situación particular, este tipo de enunciados no se pueden interpretar correctamente ya que son inherentemente ambiguos [35].

4.3.2 Vaguedad

Otro concepto que tiene relación con la ambigüedad es la *vaguedad*. Al igual que la ambigüedad, la vaguedad puede dificultar determinar la referencia al verdadero objeto del que se está hablando. No obstante, la vaguedad no provoca múltiples interpretaciones. Por ejemplo, la declaración *I want to eat Italian food* provee suficiente información quizá para recomendar un restaurante o inferir alguna inclinación de la persona hacia ese tipo de comida pero no estipula exactamente qué alimento quiere comer. La vaguedad no es inconveniente en sí misma ya que las declaraciones vagas y las concisas tienen distintos usos y son apropiadas en diferentes situaciones por lo que se debe tener mecanismos que soporten en análisis en ambos casos [37].

4.3.3 Anáfora

En lenguaje natural existen dos formas de hacer referencia a un objeto dentro de una oración: anafóricamente y no anafóricamente. La anáfora expresa una referencia a algún sustantivo o frase nominal que se ha enunciado con anterioridad en el discurso, usando normalmente un pronombre. Por ejemplo, en la frase *Jack saw himself in the mirror.* La palabra *himself* hace una referencia al sustantivo *Jack* previamente declarado. Asimismo, en las frases *Jack went to the party. He got drunk.* El pronombre *He* en la segunda oración hace referencia a *Jack* declarado en el primer enunciado.

No todas las referencias anafóricas son tan explícitas. Por ejemplo, una frase nominal puede presentar un conjunto vago de objetos hacia los cuales después se hace referencia. Por ejemplo, en *We bought two guitars at the store. The new one was cheap and the used one was expensive.* La frase nominal *two guitars* presenta dos objetos sin definirlos explícitamente y la segunda oración hace referencias anafóricas a estos objetos antes mencionados. Otras formas de anáfora pueden usar referencias a eventos implícitos de enunciados anteriores. Por ejemplo, *Jack congratulated the winner. After some hesitation, John dit it too.* El pronombre *it* en el segundo enunciado hace referencia al evento de felicitar al ganador que implícitamente se define por la acción que ejecutó Jack [35].

4.4 Interpretación de enunciados basada en aterrizaje de roles semánticos

Un rol semántico se refiere a una frase nominal que cumple un propósito específico respecto a la acción o al estado que describe el verbo principal del enunciado. La descripción completa del evento se puede modelar como una función con parámetros que corresponden a roles semánticos en el evento que describe el verbo, como pueden ser *actor, objeto, instrumento, lugar destino, tiempo de inicio*, etc. Existe una gran varie-

dad de combinaciones posibles de verbos y de parámetros que deben localizarse en el enunciado. Estos roles no siempre se presentan verbalmente por lo que se deben inferir del contexto, resolverse con un valor por defecto o entablar una interacción verbal adicional con el usuario para clarificar el significado. Asociar un fragmento del enunciado a algún parámetro que modifica la descripción del evento no es directo ya que cada parámetro puede estar sujeto a restricciones semánticas expresadas en el modelo del mundo. Considérese el verbo *eat*, con los parámetros de *actor, objeto y lugar*. En la oración *Anna eats a sandwich*, *Anna* es el actor, la frase nominal *a sandwich* es el objeto, y el lugar se debe consultar del modelo del mundo como la ubicación conocida de *Anna*. Si la oración es *A sandwich, Anna eats* la interpretación debe permanecer igual a pesar que el orden de aparición cambió. Por lo tanto, se debe usar el modelo del mundo para verificar que los objetos asociados con los roles del verbo tengan la intención original.

La interpretación de una variedad de enunciados que describen un solo evento se puede plantear usando fragmentos de la frase que tienen relación con algún parámetro que describe el evento. En este trabajo propongo patrones de interpretación que enumeran tanto a los fragmentos del enunciado asociados a eventos y roles semánticos como de las funciones que generan las expresiones de significado, comandos e interacciones verbales. En consecuencia, el lenguaje aceptado por el intérprete depende, en su nivel de abstracción más alto, del conjunto de patrones que definen la asignación de significados y comandos al planeador de acciones.

4.4.1 Patrones de interpretación

En esta tesis, la especificación sobre el proceso de traducción desde un lenguaje natural controlado a distintos lenguajes de significado se expresa mediante un patrón como el que se muestra en el cuadro 4.1.

Dependencia conceptual	funcion_gen_dep(param1, ..., paramn)			
Confirmación verbal	funcion_gen_conf(param1, ..., paramn)			
Comando al planeador +	funcion_gen_com1(param1, ..., paramn)			
Comando al planeador -	funcion_gen_com2(param1, ..., paramn)			
Parámetros	**Constituyente**	**Tipos semánticos**	**Palabras clave**	**Valor en ausencia de especificación**
param1	consituyente_1	tipo_11, ..., tipo_1k	palabra_11, ..., palabra_1j	funcion_externa_1(A, B, ...)
paramn	consituyente_n	tipo_n1, ..., tipo_nk	palabra_n1, ..., palabra_nj	funcion_externa_n(A, B, ...)

Cuadro 4.1 Patrón genérico de interpretación de verbos.

El patrón de interpretación consta de expresiones de salida que usan los roles semánticos del verbo y de la descripción de tales roles. La descripción de los roles se compone de información sintáctica al definir de qué constituyentes puede provenir la instancia y de información semántica, al enunciar las categorías de objetos compatibles. Además, la descripción incluye una lista de palabras clave que deben estar contenidas en el constituyente y, en caso de que ese rol no se presente en el enunciado, un valor por defecto o una función que consulta a la base de conocimientos o a agentes externos para identificar el objeto que cumple ese rol semántico. Durante el proceso de interpretación de la oración se genera una lista de pares rol-instancia que expresa qué objeto en la ontología cumple el rol del actor, objeto, tiempo, etc. En el cuadro 4.1 se muestra el patrón genérico de interpretación. Para facilitar el diseño de interfaces humano-computadora con base en la interpretación de la oración, el sistema genera las siguientes cuatro expresiones:

- **Dependencia conceptual**. Expresa el significado no lingüístico del enunciado. Por ejemplo, la oración *Peter moves the blue chair to the living room* tiene el significado en dependencias conceptuales *(PTRANS (ACTOR peter) (OBJECT my_chair) (ORIGIN: kitchen) (DESTINATION: livingroom))*

- **Confirmación verbal**. Es una paráfrasis en lenguaje natural que se repite al usuario para confirmar que se ha interpretado como el hablante pretendió.

- **Comando al planeador +**. En caso que el usuario confirme la correcta interpretación de la oración, se envía esta expresión para ser evaluada en el planeador de acciones.

- **Comando al planeador —**. En caso que el usuario rechace la interpretación propuesta se envía esta expresión al planeador.

Por ejemplo, el verbo *bring* con los parámetros *actor, objeto, lugar inicio y lugar destino* abarcaría oraciones como *anna brings the green chair from the kitchen to the livingroom*. El *actor* este verbo debe ser un objeto animado normalmente de la clase *person* o *robot* y esta instancia se espera que se describa dentro de una frase nominal. Para este ejemplo, el *objeto* debe ser una instancia de *furniture, food o drink* descrita también en una frase nominal y no tiene valor por defecto. Por ultimo, los roles *lugar inicio* y *lugar destino* se deben expresar en frases preposicionales que incluyan una frase nominal que refiera a un objeto de la clase *place* y que tengan las palabras clave *from* y *to*, respectivamente. Si esas frases preposicionales no se incluyen en la oración, entonces *lugar inicio* es la última ubicación conocida del objeto y *lugar destino* es la ubicación actual del robot. Con lo anterior se construye el patrón del cuadro 4.2.

La especificación del significado de las oraciones que usan un verbo requiere múltiples patrones como el anterior para cubrir todos los casos de interés en la aplicación.

Dependencia conceptual	(PTRANS (ACTOR actor) (OBJECT objeto) (ORIGIN: lugar_inicio) (DESTINATION: lugar_destino))			
confirmación verbal	ok actor moves object to lugar_destino			
Comando al planeador +	(ACTION: bring, ACTOR: actor, OBJECT: objeto, ORIGIN: lugar_inicio, DESTINATION: lugar_destino)			
Comando al planeador -	(ACTION: say, MESSAGE: sorry try to rephrase the sentence)			
Parámetros	**Constituyente**	**Tipos semánticos**	**Palabras clave**	**Fuente externa**
actor	frase nominal	person, robot	-	-
objeto	frase nominal	furniture, drink, food	-	-
lugar inicio	frase preposicional	place, person	from	consult_kb(location, objeto)
lugar destino	frase preposicional	place, person	to	consult_kb(location, robot)

Cuadro 4.2 Patrón de interpretación del verbo bring.

4.4.2 Relación con la base de conocimiento

La ontología almacena como nombres propios a todas las categorías e instancias de objetos, lugares, etc. Las instancias de las categorías de atributos se manejan como adjetivos en lenguaje natural. El sistema mantiene un vocabulario finito, por lo que las palabras que pertenecen a una categoría gramatical abierta deben estar declaradas en la ontología. Asimismo, las palabras de categorías cerradas se codifican en la misma ontología con los vértices especiales del tipo *pos(the, determiner), pos(where, adverb,), pos(from, preposition), pos(drink, verb)*, etc.

Uno de los problemas comunes al mezclar la ontología con el léxico es que la ontología almacena la *raíz* de la palabra, es decir su forma básica sin inflexiones ni declinaciones. Esto introduce la necesidad de rutinas de mapeo de palabras morfológicamente alteradas a la raíz de la palabra para poder ligarla con nodos de la ontología. En este trabajo considero únicamente rutinas para identificar sufijos de plurales y palabras compuestas. Por lo que *chairs* se identifica con el nodo *chair* y *kitchen table* se identifica con el nodo *kitchen_table*. Además, se pueden declarar sobrenombres a cualquier nodo en la ontología para forzar que dos o más palabras se traten por igual.

4.4.3 Conexión con el planeador automático

Una vez que el enunciado se interpreta, se clasifica para saber que tipo de acción tomar enseguida. Si la entrada corresponde a una orden al robot, entonces el módulo

de interpretación de lenguaje debe proveer la información necesaria en un formato adecuado para iniciar el proceso de planeación. El proceso de interpretación de la oración debe producir una expresión que describa de qué forma seleccionar y inicializar el primer nodo del grafo que se va a explorar en el proceso de planeación. Como el planeador tiene acceso directo al mismo estado del mundo que el módulo de lenguaje, la expresión únicamente debe especificar los parámetros aterrizados del nodo del grafo de tareas no primitivas donde inicia el proceso de planeación. La descripción del nodo es de la forma *(METODO_K parámetro_1:valor_1, parámetro_2:valor_2 ...).*

4.5 Proceso de interpretación de un enunciado

En el contexto de este trabajo, interpretar una oración es asociar el texto de entrada a un patrón de interpretación cuyos roles semánticos están ligados a un nodo de la ontología.

La interpretación del enunciado consta de tres pasos: generación de metadatos de palabras y constituyentes, asociación de roles y generación de expresiones de salida. Los metadatos contienen la información necesaria para asociar cada palabra o constituyente con un rol semántico.

4.5.1 Generación de metadatos de palabras y constituyentes

Los metadatos que acompañan a cada palabra o constituyente constan de una tabla con cuatro campos: *constituyente o categoría gramatical, tipos semánticos, palabras clave e instancias resueltas.* Extraer esa información del texto requiere análisis sintáctico, semántico y consultas al estado del mundo. En el cuadro 4.3 se muestra el esquema genérico de la asociación del enunciado etiquetado a metadatos.

Enunciado de entrada	word_1 word_2 ... word_n			
Palabra o constituyente	**Constituyente o categoría gramatical**	**Tipos semánticos**	**Palabras clave**	**instancias resueltas**
word_1-word_i	constituent_1 o POS_1	superclass_1, ..., top	word_1-word_i	object_a, ..., object_b
word_i+1-word_j	constituent_2 o POS_2	superclass_2, ..., top	word_i+1-word_j	object_c, ..., object_d
...	...	...	...	
word_k-word_n	constituent_m o POS_m	superclass_m, ..., top	word_k-word_n	object_y, ..., object_z

Cuadro 4.3 Enunciado genérico etiquetado con metadatos

Se comienza con etiquetar cada palabra del enunciado de entrada con su categoría gramatical usando los nodos de la ontología *pos(palabra, categoría)*. El subárbol de la ontología a partir de *stuff* son sustantivos y a partir de *attribute* son adjetivos. Es frecuente encontrar que múltiples categorías gramaticales se pueden asignar a una palabra, provocando así *ambigüedad léxica* en el etiquetado de categorías gramaticales o *pos tagging*. Para resolverlo, el sistema usa *desambiguación léxica basada en reglas* que consta de un conjunto de reglas del tipo *una preposición precede a un determinante* y *un adjetivo precede a un sustantivo*. Además, se tienen reglas que incluyen información sobre las categorías de la ontología y son de la forma *una preposición de lugar precede a un sustantivo de la categoría place*. Estas reglas sirven para determinar cuál es el correcto etiquetado considerando todas las posibles combinaciones de etiquetas. El seudocódigo se presenta en el algoritmo 1:

```
Data: input_sentence, ontology, disambiguation_rules
list_of_pos ← ∅
for each_word in input_sentence do
 |  list_of_pos.Append( get_list_of_postags(each_word)
end
all_combinations ← get_all_combinations(list_of_pos)
ranked_combinations ← rank_combinations(all_combinations, disambiguation_rules)
return ranked_combinations.FirstItem
```

Algorithm 1: Algoritmo de desambiguación léxica.

El algoritmo primero asocia cada palabra a una lista de posibles categorías gramaticales en la lista de listas *list_of_pos*. Con esa lista se generan todas las posibles combinaciones de pares palabra-categoría para etiquetar el enunciado en la lista de listas *all_combinations*. Usando las reglas de desambiguación, se asigna un puntaje a cada posible combinación de etiquetas gramaticales y se regresa la combinación mejor calificada.

El siguiente paso es agrupar las palabras en constituyentes como frases nominales o frases preposicionales. Se usa análisis sintáctico superficial o *shallow parsing* para implementar un trozador o *chunker* que separe estructuras sintácticas importantes que están dentro de la oración y que normalmente comprenden únicamente a un subconjunto de las palabras en la oración. Para localizar y separar cada constituyente, el trozador usa gramáticas libres de contexto y un verificador de gramáticas CFG para encontrar exhaustivamente los constituyentes más extensos que cumplan con la gramática proveída. Para este trabajo únicamente se localizan y separan frases nominales y preposicionales. No obstante, la misma idea se puede extender a otros constituyentes.

En lenguaje natural se suele hacer referencia a instancias y categorías mediante descripciones como *the cleanest blue chair*. Sin embargo, para usarse como un parámetro en una función de localización, reconocimiento, manipulación, etc, el objeto requiere especificarse mediante un nodo de la ontología o por una expresión válida para el planeador de acciones. Por ello, se necesita que las frases nominales se *aterricen* a un

```
Data: input_sentence, grammar_g
list_of_chunks ← ∅
for start in (1, input_sentence.Number_of_words) do
    best_chunk ← (0,0)
    for end in (start, input_sentence.Number_of_words) do
        subset_checked ← verify_cfg(input_sentence.words[start, end], grammar_g)
        if subset_checked == True then
            best_chunk = (start, end)
        end
    end
    if best_chunk != (0,0) then
        list_of_chunks.Append(best_chunk)
        start ← end
    end
end
return list_of_chunks
```

Algorithm 2: Algoritmo de identificación de constituyentes de la oración usando CFG.

subconjunto de instancias en el modelo del mundo o se generen expresiones válidas de *evaluación perezosa*. En el algoritmo 3 se explica el método de aterrizaje de frases nominales usado en este trabajo.

```
Data: input_noun_phrase, ontology
list_of_intances ← ∅
main_noun ← get_main_noun(input_noun_phrase)
list_of_attributes ← get_attributes(input_noun_phrase)
list_of_values ← get_values(input_noun_phrase)
quantifier ← get_quantifier(input_noun_phrase)
list_of_candidate_intances ← get_inherited_instances(main_noun)
for each_candidate in list_of_candidate_intances do
    include_instance ← check_constrains(each_candidate, list_of_attributes, list_of_values +
    list_of_secundary_nouns)
    if include_instance == True then
        list_of_intances.Append(each_candidate)
    end
end
if list_of_instances == ∅ then
    return generate_lazy_evaluation_expression(main_noun, list_of_attributes, list_of_values,
    list_of_secundary_nouns)
else
    return get_subset(list_of_instances, quantifier)
end
```

Algorithm 3: Algoritmo de aterrizaje de frases nominales.

Cuando una frase nominal entra al algoritmo de aterrizaje, existen dos posibles casos que pueden ocurrir: si el algoritmo logra aterrizar la frase nominal a un conjunto de nodos en la ontología, regresa el nombre de los objetos encontrados. Si el algoritmo no encuentra tales nodos en la ontología, se genera una expresión de *evaluación perezosa*

para que el planeador de acciones intente aterrizar ese parámetro en otro momento. Este algoritmo comienza con la identificación de la clase principal de la frase nominal, las clases de atributos, los adjetivos y los cuantificadores. Por ejemplo, en *two cold drinks* la clase principal es *drink*, *two* es un cuantificador y *cold* un adjetivo. Algunas veces otros sustantivos secundarios en la frase nominal cumplen como valores de atributos, por ejemplo en *the house in germany*, *germany* es un valor de ubicación que califica a alguna instancia de *house*. Una vez obtenidos la clase principal de la frase nominal, las clases de atributos, los adjetivos y los cuantificadores, se extraen todas las instancias de la clase principal. Usando ese conjunto de instancias candidatas y las clases de atributos y adjetivos, se localizan los objetos en la ontología que cumplen con la descripción expresada en la frase nominal. Finalmente, usando el cuantificador de la frase nominal se decide cuántas instancias se regresan como resultado del algoritmo.

Si el algoritmo no logra aterrizar la frase nominal, genera una expresión que contenga la información necesaria para que el planeador pueda intentar aterrizar la frase nominal bajo demanda usando *evaluación perezosa*.

Por ejemplo, el enunciado *anna brings the blue chair from the kitchen* produciría un patrón como en el cuadro 4.4.

Enunciado de entrada	anna brings the blue chair from the kitchen			
Palabra o constituyente	**Constituyente o categoría gramatical**	**Tipos semánticos**	**Palabras clave**	**instancias resueltas**
anna	frase nominal	woman, person, top	anna	anna
brings	verbo	action, top	brings	-
the blue chair	frase nominal	chair, furniture, stuff, top	blue, chair	my_chair_x
from the kitchen	frase preposicional	room, place, stuff, top	from, kitchen	this_kitchen_x

Cuadro 4.4 Enunciado etiquetado con metadatos.

4.5.2 Asociación de roles

Una vez que las palabras en la oración de entrada están agrupadas y etiquetadas, se buscan patrones de interpretación que describan roles semánticos compatibles con las estructuras encontradas.

El algoritmo toma el conjunto de patrones de interpretación almacenados y busca en ellos los roles semánticos compatibles con las estructuras encontradas en la oración de entrada. El algoritmo intenta interpretar la oración con cada uno de los patrones de interpretación disponible. Cada intento de interpretación se califica con un valor de 0 a 1, dependiendo del número de roles semánticos asociados y con el número de palabras

```
Data: sentence_metadata, interpretaion_pattern_set
list_of_interpretations ← ∅
for each_pattern in interpretaion_pattern_set do
    for each_parameter in sentence_metadata.parameters do
        for each_parameter_to_asociate in each_pattern.parameters do
            each_parameter_to_asociate ← try_asociate(each_parameter_to_asociate,
            each_parameter)
            if try_asociate(each_parameter_to_asociate, each_parameter) succeded then
                interpretation.solved_parameters.append(each_parameter_to_asociate,
                each_parameter)
            end
        end
    end
    interpretation.rank ← get_value(sentence_metadata.parameters,
    interpretation.solved_parameters) list_of_interpretations.append(interpretation)
end
ranked_list ← sort_by_rank(list _of_interpretations)
if ranked_list.first.rank ≥ threshold then
    return ranked_list.first
else
    return ∅
end
return ranked_list.first
```

Algorithm 4: Algoritmo asociación de los metadatos de un enunciado a una interpretación.

en la oración que no se utilizan. Si un patrón de interpretación logra satisfacer todos los roles semánticos estipulados y para ello se usan todas las palabras en la oración de entrada, su calificación será 1, lo que significa que el patrón describe exactamente la estructura y tipos de objetos que exhibe el enunciado. Los patrones que no interpretan adecuadamente al enunciado de entrada tendrán roles semánticos que no se pueden asociar y existirán palabras en el enunciado se ignoren, lo que causa que tengan una calificación baja o nula. El algoritmo regresa el patrón aterrizado que mayor porcentaje de roles logró asociar si cumple con un umbral especificado.

Capítulo 5

Planeación automática

La *planeación automatizada* es el área de la inteligencia artificial que estudia el proceso de elección y organización acciones mediante la anticipación de sus efectos para cumplir objetivos definidos.

Algunas acciones requieren de planeación, pero otras no. En la vida cotidiana se realizan a diario muchas acciones que no necesitan un proceso de deliberación explícito y no siempre se tiene presente sus efectos de forma anticipada, es decir, no siempre se requiere planear antes de actuar. No siempre es necesario planear cuando el propósito de la acción es inmediato al acto, cuando se ejecuta una actividad para la cual se está bien entrenado y cuando la acción se puede adaptar libremente al tiempo que se ejecuta. La ejecución de una actividad requiere planeación cuando se intenta resolver una tarea con planteamiento u objetivos complejos y cuando no se está familiarizado con las acciones que resuelven la actividad. Asimismo, se necesita planeación cuando la selección de las tareas a resolver están supeditadas a cambios en el ambiente que implican un alto riesgo o alto costo.

La planeación automatizada es importante en el estudio y diseño de máquinas autónomas como satélites y robots domésticos que ahora se benefician de un comportamiento autónomo gracias a este tipo de deliberación artificial, resolviendo problemas en situaciones donde no pueden operarse a distancia por un humano o donde la aplicación así lo requiere [39].

Levesque y Reiter han hecho las siguientes preguntas para dar un planteamiento al problema [21]:

- Para ejecutar un programa (o plan de acción) ¿Cuál es la información que el robot debe tener en un principio y qué información necesita adquirir en tiempo de ejecución mediante mecanismos de percepción?
- ¿Qué debe saber el robot sobre su ambiente y qué debe conocerse únicamente por

su diseñador?

- ¿Cuándo es necesario que el robot use mecanismos de percepción para verificar si algo es verdadero y cuándo debe inferirlo de conocimiento almacenado con anterioridad?

- ¿Cuándo debe darse una descripción detallada de una actividad para que el robot pueda razonar sobre ella y cuándo una actividad debe considerarse atómica o primitiva?

Ninguna de estas preguntas tiene un clara respuesta porque depende en gran medida de las capacidades y uso del robot en cuestión.

5.1 Conceptos básicos de planeación automatizada

Para cualquier algoritmo de planeación se necesita una descripción formal del problema que se requiere resolver. Esta descripción debe contener de manera implícita o explícita las distintas configuraciones o estados que el mundo puede presentar durante la ejecución de acciones. Debido a la complejidad del modelo del mundo, para muchos dominios es imposible enumerar explícitamente cada una de las configuraciones posibles a las que se puede llegar mediante una secuencia de acciones. El lenguaje de representación debe diseñarse de tal forma que sea fácil computar tales estados *al vuelo* [39].

5.1.1 Representación del plan

En la representación conjuntista cada *estado* del mundo es un conjunto de proposiciones y cada *acción* es una expresión que especifica tanto las proposiciones necesarias para aplicar esa acción como las proposiciones que se volverán verdaderas y falsas después de haber finalizado la aplicación de la acción en cuestión.

Sea $L = p_1, \ldots, p_n$ un conjunto finito de variables proposicionales. Un dominio de planeación conjuntista sobre L es un sistema de estado-transición $\Sigma = (S, A, \gamma)$ tal que [39]:

- Cada estado s es un subconjunto de L. s declara cuáles son las proposiciones verdaderas en la representación del mundo. Si $p \notin s$ entonces p no se considera verdadera en la representación del mundo que expresa s. S es el conjunto de todas las configuraciones válidas en la representación del mundo.

- Cada acción $a \in A$ es una terna de subconjuntos de L, del tipo $a = (precondiciones(a), efectos^{-}(a), efectos^{+}(a))$, donde $efectos^{-}(a)$ y $efectos^{+}(a)$ son conjuntos disjuntos. Se dice que la acción a es *aplicable* en el estado s si $precondiciones(a) \in s$. Al aplicar a en s se llega al estado s', donde $efectos^{-}(a) \notin s'$ y $efectos^{+}(a) \in s'$.

- s tiene la propiedad que si $s \in S$, entonces para cada acción a aplicable en s, el conjunto $efectos^{-}(a) \cup efectos^{+}(a) \in S$. Es decir, aplicar una acción en un estado válido produce necesariamente otro estado válido.

- La función de estado-transición es $\gamma(s, a) = ((s - efectos^{-}(a)) \cup efectos^{+}(a))$ siempre que $a \in A$ es aplicable en $s \in S$, de lo contrario la función está indefinida

Un *problema de planeación* se especifica semánticamente por una terna $P = (\Sigma, s_0, s_f)$, donde:

- s_0 es el estado inicial $s_0 \in S$,

- $f \in L$ son un conjunto de proposiciones meta que dan los requisitos del estado meta $s_f \in S$, y

- Σ es el dominio de planeación $\Sigma = (S, A, \gamma)$ independiente de cualquier estado inicial u objetivo, donde S es el conjunto de configuraciones válidas del mundo, A es el conjunto de acciones de manipulación del mundo y γ es el conjunto de funciones de estado-transición de cada acción.

Un *plan* es una secuencia de acciones $\pi = \langle a_1, ..., a_k \rangle$ donde $k \geq 0$. La *longitud* del plan es $|\pi| = k$ el número de acciones.

El estado producto de aplicar un plan π es el estado al que se llega luego de aplicar las acciones contenidas en π en el orden dado. Esto se denota extendiendo la definición de la función de estado-transición de forma recursiva [39]:

$$\gamma(s, \pi) = \begin{cases} s & si \quad k = 0 \quad (plan \ \ vacio) \\ \gamma(\gamma(s, a_1), \langle a_2, \ldots, a_k \rangle) & si \quad k > 0 \quad a_1 \ \ aplicable \ \ en \ \ s \end{cases} \tag{5.1}$$

Sea $P = (\Sigma, s_0, s_f)$ un problema de planeación. El plan π es una solución de P si $s_f \subset \gamma(s_0, \pi)$. Se dice que π es una solución redundante si existe un fragmento π' del plan π tal que $|\pi'| < |\pi|$ y donde π' también es una solución a P. π es mínimo si no existe otra solución a P con menos acciones que π [39].

5.2 Planeadores de redes jerárquicas de tareas HTN

Los planeadores de redes jerárquicas de tareas o *Hierarquical Task Network* (HTN) difieren de otros tipos de planeadores debido a que no buscan explícitamente alcanzar un estado objetivo, sino ejecutar un conjunto de *tareas*. La entrada al sistema de planeación incluye un conjunto de operadores similar a las *acciones* en la planeación clásica llamados *métodos*. Cada método es una descripción de cómo descomponer una *tarea no primitiva* recursivamente en un conjunto de *subtareas*. La descomposición de subtareas acaba cuando únicamente se tienen *tareas primitivas* que se pueden ejecutar directamente en el sistema como un operador de planeación [39].

La planeación usando redes jerárquicas de tareas provee una manera de describir patrones de la solución de problemas que corresponden a la manera en la que actuaria un experto humano.

5.2.1 Planeación de redes de tareas simples

Una versión simplificada del planeador HTN son las redes de tareas simples o *Simple Tasks Network* (STN). En esta simplificación las definiciones de literales, operadores, acciones y plan son las mismas que en la planeación clásica. Además, este modelo incluye *tareas*, *métodos* y *redes de tareas* como elementos nuevos.

Una red de tareas es un grafo acíclico dirigido $w = (U, E)$ donde U es el conjunto de nodos y E el conjunto de aristas. Cada nodo $u \in U$ contiene una tarea t_u. Se dice que w está aterrizada si todas las tareas $t_u | u \in U$ están aterrizadas. Asimismo, se dice que w es primitiva si todas las t_u son primitivas. Las aristas E del grafo proveen un ordenamiento parcial de las tareas, $u \prec v$ si existe un camino de u hacia v.

Un *método* es una cuádrupla $m = (nombre(m), tarea(m), precondiciones(m), red(m))$ donde [39]:

- **Nombre**. Es el nombre del método con la sintaxis $n(x_1, \ldots, x_k)$ donde n es un símbolo único de identificación del método y $x_1, \ldots, x_k$ son todas las variables que acontecen en m.
- **Tarea**. Es una tarea no primitiva que indica que tipo de tarea puede aplicar a m como sub-tarea.
- **Precondiciones**. De manera similar a las tareas, el conjunto de literales de precondiciones son hechos que se deben cumplir para que el método sea aplicable en un estado dado.
- **Red**. Es una red de tareas simples cuyas tareas se les llama sub-tareas de m.

Si t es una tarea, m una instancia de un método y existe una sustitución de σ del tipo $\sigma(t) = tarea(m)$, entonces m es relevante para t y la descomposición de t usando m mediante σ es $\delta(t, m, \sigma) = red(m)$. Es decir, se usa a la red de m para describir de forma más explícita o descomponer a una porción de t. Cuando se usa un método m para descomponer una tarea t normalmente esa tarea forma una parte de un nodo u dentro de otra red w, en cuyo caso después de la descomposición se tiene un nuevo grafo donde el nodo u se ve reemplazado por $red(m)$, donde cada restricción de orden parcial que aplicaba a u ahora aplica a $red(m)$.

Un problema de planeación STN queda definido por una terna $P = (s_0, w, \Sigma)$, donde s_0 es la configuración inicial del mundo, w es una red de tareas iniciales y Σ es el dominio de planeación. Nótese que no se tiene explícitamente una configuración objetivo sobre el mundo.

Se dice que $\pi = \langle a_1, \ldots a_n \rangle$ es una solución de P si existe una forma de descomposición de w en π tal que cada descomposición es aplicable en cada configuración del mundo apropiadamente. La definición formal se descompone en tres casos recursivos [39]:

- w está vacía. Entonces π es una solución de P, $n = 0$.

- Existe un nodo de una tarea primitiva u en w tal que no tiene predecesores. Entonces π es una solución de P si a_1 es aplicable a t_u en s_0 y el plan restante $\pi' = \langle a_2, \ldots a_n \rangle$ resuelve al problema restante $P' = (\gamma(s_0, a_1), w - u, \Sigma)$. Es decir, si existe un nodo de una tarea lista para ser resuelta mediante una tarea primitiva, la tarea se debe ejecutar y retirar el nodo correspondiente del grafo.

- Existe un nodo de una tarea no primitiva u en w tal que no tiene predecesores. Si existe un método m tal que es relevante para t_u en s_0, entonces π es una solución de P si existe una red de tareas $w' = \delta(w, u, \Sigma)$ tal que π es una solución para $P' = (s_0, w', \Sigma)$. Es decir, si la tarea que se debe ejecutar es no primitiva y es relevante en s_0, se sustituye u por $red(t_u)$ para conformar w' y el problema P se formula como P'.

Si π es solución de P, entonces para cada nodo $u \in w$ existe un árbol de descomposición tal que las hojas del árbol corresponden a las acciones en π. Si el nodo u corresponde a una acción primitiva entonces t_u es el nombre de la acción que lo soluciona en π y su árbol de descomposición sólo contiene u. Si el nodo u corresponde a una acción no primitiva entonces la descomposición corresponde al grafo resultado de la operación $\delta(w, u, \Sigma)$.

5.2.2 Planeación totalmente ordenada contra parcialmente ordenada

El método *descomposicón totalmente ordenada hacia adelante* o *total-order forward decomposition* TFD, cuyo seudo-código se presenta en el algoritmo 5, sirve para descomponer redes jerárquicas de tareas tales que cada nodo tiene al menos un nodo antecesor [39].

Data: s,π, Σ
if $k = 0$ **then**
  return $\langle\rangle$ (el plan vacío)
end
if t_1 *es una tarea primitiva* **then**
  active $\leftarrow (a, \sigma)$ tal que a es una instancia aterrizada de un operador en $\Sigma = (O, M)$, $a \in O$. σ es una sustitución $\sigma \in \Sigma$ tal que a es relevante para $\sigma(t_1)$ y a es aplicable en s.
  if *active* $= \emptyset$ **then**
    regresar falla (no hay una tarea primitiva ejecutable)
  end
  Elegir de manera no determinística algún $(a, \sigma) \in$ *active*
  $\pi \leftarrow TFD(\gamma(s, a), \sigma(\langle t_2, ..., t_k\rangle), \Sigma)$
  if π *falla* **then**
    regresar falla
  else
    regresar a y π
  end
else if t_1 *es una tarea no primitiva* **then**
  active $\leftarrow$ (m tal que m es una instancia aterrizada de un método en $\Sigma = (O, M)$, $m \in M$. σ es una sustitución $\sigma \in \Sigma$ tal que m es relevante para $\sigma(t_1)$ y m es aplicable en s.
  if *active* $= \emptyset$ **then**
    regresar falla (no hay un método para aplicar una descomposición de tareas)
  end
  Elegir de manera no determinística algún $(m, \sigma) \in$ *active*
  $w \leftarrow subtareas(m).\sigma(\langle t_2, ..., t_k\rangle)$
  regresar TDF(s,w, Σ)

Algorithm 5: Algoritmo TFD.

El algoritmo TDF:

- Únicamente considera tareas cuyas precondiciones se satisfacen en el estado presente y que son relevantes para la red de tareas que se desea ejecutar.

- Agrega acciones al plan en orden en que se deben ejecutar. Es decir, cuando el planeador elige una tarea a ejecutar ya ha elegido las acciones previas.

En contraste, cuando la red de tareas no está totalmente ordenada pueden existir múltiples nodos relevantes en un estado del mundo, por lo que existe una decisión no determinista sobre a cuál nodo corresponde la siguiente sustitución. El algoritmo se modifica como se describe en el algoritmo 6 [39].

Data: s,π, Σ
if $w = \emptyset$ **then**
  return $\langle\rangle$ (el plan vacío)
end
de manera no determinística elegir un nodo $u \in w$ que no tenga predecesores.
if t_u *es una tarea primitiva* **then**
  active $\leftarrow (a, \sigma)$ tal que a es una instancia aterrizada de un operador en $\Sigma = (O, M)$, $a \in O$. σ es una sustitución $\sigma \in \Sigma$ tal que a es relevante para $\sigma(t_u)$ y a es aplicable en s.
  if *active* $= \emptyset$ **then**
    regresar falla (no hay una tarea primitiva ejecutable)
  end
  Elegir de manera no determinística algún $(a, \sigma) \in$ *active*
  $\pi \leftarrow PFD(\gamma(s, a), \sigma(\langle t_2, ..., t_k \rangle), \Sigma)$
  if π *falla* **then**
    regresar falla
  else
    regresar a y π
  end
else if t_1 *es una tarea no primitiva* **then**
  active $\leftarrow$ (m tal que m es una instancia aterrizada de un método en $\Sigma = (O, M)$, $m \in M$. σ es una sustitución $\sigma \in \Sigma$ tal que m es relevante para $\sigma(t_u)$ y m es aplicable en s.
  if *active* $= \emptyset$ **then**
    regresar falla (no hay un método para aplicar una descomposición de tareas)
  end
  Elegir de manera no determinística algún $(m, \sigma) \in$ *active*
  Elegir de manera no determinística cualquier red de tareas $w' \in \delta(w, u, \Sigma)$
  regresar PDF(s,w', Σ)

Algorithm 6: Algoritmo PFD.

5.3 Planeación en un mundo abierto usando rutinas de percepción

Planear acciones futuras es especialmente difícil para agentes que operan en un mundo abierto y dinámico donde no se dispone de una representación del mundo actualizada ni completa. La *planeación continua* permite intercalar procesos de planeación con la ejecución de tareas de adquisición de información externa. Se puede aumentar el modelo de STN de descomposición hacia adelante antes descrito para incluir tales tareas de adquisición de información. El algoritmo genera el plan mediante la sucesiva elección de un método o tarea relevante a la configuración del mundo en un momento dado. Sin embargo, en dominios abiertos sería posible que métodos o tareas se vuelvan relevantes si se pudiera acceder a información adicional durante el proceso de planeación. Para recabar esa información se ha propuesto una nueva clase de operadores de planeación llamados *métodos STN posiblemente aplicables* [40].

Un planeador STN no puede continuar el proceso de planeación cuando no existen precondiciones que vuelvan a una tarea relevante. Es decir, la secuencia de descomposición con operadores de planeación se detiene al no tener el contexto para provocar una sustitución adecuada. Sin embargo, si fuera posible obtener información adicional para hacer relevante una tarea antes irrelevante, sería posible continuar el proceso de planeación.

Se definen dos nuevos conceptos: los *hechos posiblemente derivables* (precondiciones) y las *literales abiertas*. Si L_x es el conjunto de literales y p es una precondición, se dice que p es posiblemente derivable respecto a L_x si y solo si la existencia de una instancia aterrizada de L_x implica la existencia de p. La dependencia entre las literales abiertas necesita ser considerada en los planes de generación de conocimiento. Por ejemplo, para el conjunto de literales abiertas $objecto(X), color(X, rojo)$ no se puede conseguir de manera independiente una instancia de $objeto(X)$ y de $color(X, rojo)$ porque se requiere que la instancia cumpla con las literales de forma simultánea. [40].

Los robots usualmente obtienen información de distintas fuentes externas por lo que tienen que efectuar consultas o comandos sobre otras bases de conocimiento o módulos de comportamientos y percepción. En particular, para encontrar una tarea que ayude al algoritmo de planeación en un mundo abierto son de interés las *tareas de adquisición de conocimiento*. Una tarea de adquisición de conocimiento tiene la forma $tac(l, I, C, ks)$ donde l es una literal, I es el conjunto de instancias derivables de l, C es el conjunto de literales dependiente de l y ks es la fuente del conocimiento. Es decir, es la tarea de adquirir la instancia σ_l de ks tal que $\sigma_l \notin I$ (no es actualmente derivable) y que para todo $c \in C$ σ_c es derivable. Por ejemplo, la sintaxis $tac(orden\_bebida(X), [temperatura(X, alta)], [class\_of(X, bebida), dentro(X, cocina)], preguntar(cliente_1))$ expresa la tarea de encontrar una instancia X para la literal $orden\_bebida(X)$ tal que X no tiene temperatura alta, X es una bebida, X está dentro de la cocina y la información

se obtiene por medio de la tarea de preguntar al $cliente_1$ [40].

Una vez establecidas las tareas de adquisición de conocimiento se pueden definir los *métodos de adquisición de conocimiento*. Un método describe las precondiciones y la secuencia de acciones de percepción necesaria para recabar la información y tiene la forma $mac(tac(l, I, C, ks), precondiciones, subtareas)$. Las tareas de adquisición de conocimiento permiten al planeador elegir una tarea para obtener información adicional para completar el proceso de planeación [40].

Algunos planeadores hacen uso del concepto conocido como evaluación perezosa, lo cual es la capacidad del planeador de postergar la evaluación de expresiones hasta el momento en que sean necesarias para continuar [24].

5.4 Diseño del planeador de tareas para un robot doméstico

El planeador toma en consideración dos tipos de tareas: las que resuelven un comando dado por un usuario como *traer, agarrar o buscar*, y las que resuelven situaciones que deben atenderse de manera asíncrona en cuanto se presentan, como *localizar estación de recarga de baterías, abrir una puerta cerrada que impide el paso* y *ejecutar una rutina de adquisición de información en búsqueda de cumplir una precondición.*

El planeador de acciones representa las tareas como una red de métodos que forman un grafo dirigido. El nodo inicial se elige e inicializa con valores aterrizados de la expresión de salida del enunciado interpretado. A partir de este nodo, el planeador comienza el proceso de sustitución de redes de tareas hasta llegar a las tareas primitivas las cuales representan órdenes de alto nivel que otros módulos externos pueden ejecutar.

5.4.1 Tareas primitivas de un robot doméstico genérico

Se enumeran las tareas que se consideran primitivas y necesarias para cualquier robot de servicio genérico. La lista de tareas se intenta mantener mínima pero suficientemente comprehensiva para poder resolver las actividades típicas de un robot de servicio doméstico. Para el propósito de esta tesis se modela un robot móvil que puede autolocalizarse en una casa, que puede detectar de objetos y personas, que tiene al menos un brazo manipulador, con reconocimiento y generación de voz.

Se dividen las tareas primitivas en cuatro categorías: *Desplazamiento, Reconocimiento, Manipulación e Interacción verbal* En los cuadros 5.1, 5.2, 5.3 e 5.4 se muestran las tareas primitivas con sus parámetros, precondiciones y efectos.

Nombre	Descripción	Parámetros	Precondiciones	Efectos
Go	Navega a una zona especificada por una etiqueta en el mapa	Lugar	-	El robot cambia de ubicación
Aproach	Se acerca y alinea a un objeto o persona en el campo visual	Objeto o persona	Atención sobre el objeto	El robot está junto al objeto

Cuadro 5.1 Tareas primitivas de desplazamiento.

Nombre	Descripción	Parámetros	Precondiciones	Efectos
Reconocer	Busca en los alrededores del robot un objeto, persona o gesto especificado por su nombre o categoría	Nombre del objeto o persona	Robot junto al objeto	Atención sobre el objeto
Aprender	Busca en los alrededores del robot un objeto o persona e inicia rutinas de entrenamiento para reconocer un nuevo objeto	Nombre del objeto o persona	Robot junto al objeto	El robot puede reconocer el nuevo objeto, atención sobre el objeto

Cuadro 5.2 Tareas primitivas de reconocimiento.

Nombre	Descripción	Parámetros	Precondiciones	Efectos
Agarrar	Toma un objeto especificado por su nombre	Nombre del objeto	Atención sobre el objeto	Objeto en mano de robot
Recibir de mano	Recibe de la mano de una persona un objeto especificado por su nombre	Nombre del objeto	Atención sobre la persona que da el objeto	Objeto en mano de robot
Poner	Pone el objeto en mano sobre un lugar u objeto	Nombre del objeto en el brazo, nombre del lugar u objeto destino	Objeto en mano de robot, robot junto al lugar u objeto destino	Objeto en lugar destino
Dar en mano	Da el objeto en el brazo manipulador a una persona especificada por nombre o categoría	Nombre del objeto en el brazo, nombre de la persona destino	Objeto en mano de robot, robot junto a la persona destino	Objeto en mano de persona destino

Cuadro 5.3 Tareas primitivas de manipulación.

Nombre	Descripción	Parámetros	Precondiciones	Efectos
Escuchar	Reconoce un enunciado de una persona	Nombre de la persona	-	-
Decir	Reproduce un enunciado específico	Enunciado	-	-

Cuadro 5.4 Tareas primitivas de interacción verbal.

Tanto las precondiciones como los efectos pueden incluir otros tipos de verificación para la aplicación de la tarea, como pueden ser el buen funcionamiento de las capacidades del robot y validación de información temporal como la hora o un temporizador. Los efectos especificados en un método no son los únicos que se provocan tras su ejecución

porque cada módulo externo podría causar otros cambios en la base de conocimiento.

5.4.2 Diseño de métodos y tareas para un planeador STN

Especificar una red de tareas para el planeador del robot implica el diseño de un grafo cuyos nodos son la descripción de una tarea primitiva o no primitiva. Las tareas no primitivas pueden especificar otra red de tareas para evaluarse. En contraste, las tareas primitivas se ejecutan inmediatamente dado un contexto y parámetros adecuados mediante módulos externos al planeador de acciones. A continuación se muestra cómo especificar una tarea no primitiva. La información necesaria para la especificación de un nodo sigue el patrón que se muestra en el cuadro 6.3.

Identificador	metodo_k
Parámetros de entrada	param_e1, param_e2, ...param_ef
Precondiciones	precon_p1, precon_p2, ..., precon_pf
Efectos +	hecho_er1, hecho_er2, ..., hecho_erf
Efectos -	hecho_ea1, hecho_ea2, ..., hecho_eaf
Tarea primitiva o grafo de sustitución	tarea_primitiva_m(param_ei, ..., param_ej) o Grafo(V, E) donde V es conjunto de nodos y E conjunto de vértices de la red de tareas de sustitución

Cuadro 5.5 Descripción de un nodo en la red jerárquica de tareas.

Al definir un grafo de sustitución, los vértices son simplemente pares ordenados de indentificadores de nodos de la forma $E = \{$*(identificador_a, identificador_b), . . . , (identificador_i, identificador_j)*$\}$. Estos patrones definen las redes que componen la tarea.

5.4.3 Diseño de tareas de ejecución asíncrona

El diseño de las tareas de emergencia y de adquisición de conocimiento es idéntico al diseño de tareas no primitivas excepto que estas tareas no se inician debido a una solicitud del módulo de interpretación de lenguaje. En contraste, se inician de manera asíncrona cuando las precondiciones de un estado de emergencia se cumplen. En la descripción del grafo que describe todas las tareas, esto significa que las redes de tareas de emergencia son partes desconectadas del grafo principal. Cuando en un estado del mundo son relevantes tanto una tarea del plan solicitado como una tarea de emergencia, el intérprete que esté procesando las redes de tareas asigna prioridad la tarea que no sea de emergencia. Este tema se retomará cuando se hable de la implementación del planeador automático.

Capítulo 6

Implementación y codificación de micromundos

Durante el desarrollo de esta tesis programé un prototipo de software donde se implementaron las ideas descritas hasta ahora. Durante el proceso de planeación automática, el software simula los efectos que las acciones tienen sobre el modelo del mundo para hacer pruebas preliminares. Hacia el final del desarrollo del sistema, hice un conjunto de ramificaciones al proyecto principal para evaluarlo en tareas específicas propuestas por terceras partes. Estas ramificaciones incluyen la integración del sistema con el software de control del robot *Justina* del laboratorio de *Biorobótica* del Posgrado de Ingeniería y el procesamiento del corpus SEMEVAL 2014.

Los datos iniciales de la ontología, junto con los patrones de interpretación y los grafos de las métodos en el planeador, forman la descripción simbólica del micromundo. Esta información define la forma en que se procesan las oraciones de entrada. Para adaptar el software a una aplicación específica se modifican la ontología inicial, los patrones de interpretación de los enunciados, las expresiones que genera el proceso de aterrizaje de frases nominales y la descripción de los grafos que componen las tareas. El grado de detalle de la representación depende de los propósitos de la aplicación y el nivel de integración que se tenga con otros sistemas de software. En esta sección se muestran algunos detalles del prototipo de software y ejemplos de cómo configurar el sistema para que el sistema produzca interpretaciones adecuadas.

6.1 Descripción del software

El software se divide por su función en *servicios de la base de conocimiento, interpretación de oraciones* y *planeador automático*. Los servicios de consulta y almacenamiento forman una abstracción de los datos almacenados para que las rutinas de lenguaje ten-

gan acceso a todo el conocimiento explícito e implícito que contiene la ontología. Por otro lado, la tarea principal del software de lenguaje es procesar la oración de entrada y asignarle una interpretación con los patrones de interpretación que se tengan almacenados. Si la oración queda completamente interpretada bajo uno de esos patrones entonces se clasifica y, de ser el caso, se envía al planeador de acciones para producir un plan.

Además, el sistema se integró con el software que controla los comportamientos del robot humanoide *Justina*, mandando y recibiendo comandos mediante una arquitectura *blackboard* para controlar un robot real a partir de un plan de acción generado.

6.1.1 Organización general de los procesos de interpretación de lenguaje y planeación

La generación de planes a partir de un enunciado en lenguaje natural se divide en dos procesos que se ejecutan secuencialmente. Primero se mapea el enunciado de entrada a sus significados formales tanto en dependencias conceptuales como en comandos al planeador de acciones. Después se ejecuta el planeador automático a partir de la expresión generada y el modelo del mundo en ese momento. En la figura 6.1 se presenta una visión general de cómo se conectan las distintos módulos de software.

Cuando el sistema recibe una oración lo intenta interpretar con el conjunto de patrones de interpretación almacenado. El proceso de interpretación usa los algoritmos de análisis sintáctico y semántico que invocan servicios de análisis sintáctico y servicios de la base de conocimiento. Cuando el enunciado es una declaración de hechos o una pregunta, se resuelve usando un servicio de la base de conocimiento. Cuando es una orden de ejecución al robot, se envía al planeador automático. El planeador usa la base de conocimiento durante el proceso de planeación automática. Al no tener los módulos de percepción y manipulación del robot conectados a la misma base de conocimiento, el efecto de cada acción primitiva se simula al introducir o borrar conocimiento en la ontología, modificando, por ejemplo, la ubicación o existencia de objetos.

6.1.2 Base de conocimiento

La base de conocimiento mantiene el grafo de la ontología y provee de mecanismos de consulta para que, usando la topología y las propiedades algebraicas de las relaciones entre nodos, los procesos de interpretación de lenguaje y de planeación tengan toda la información explícita e implícita expresada en el grafo.

La base de conocimiento está programada en python. La ontología se lee a partir de archivos de texto plano que contienen expresiones del tipo (*objeto_a relacion objeto_b*) que describen un vértice etiquetado. Sus funciones incluyen:

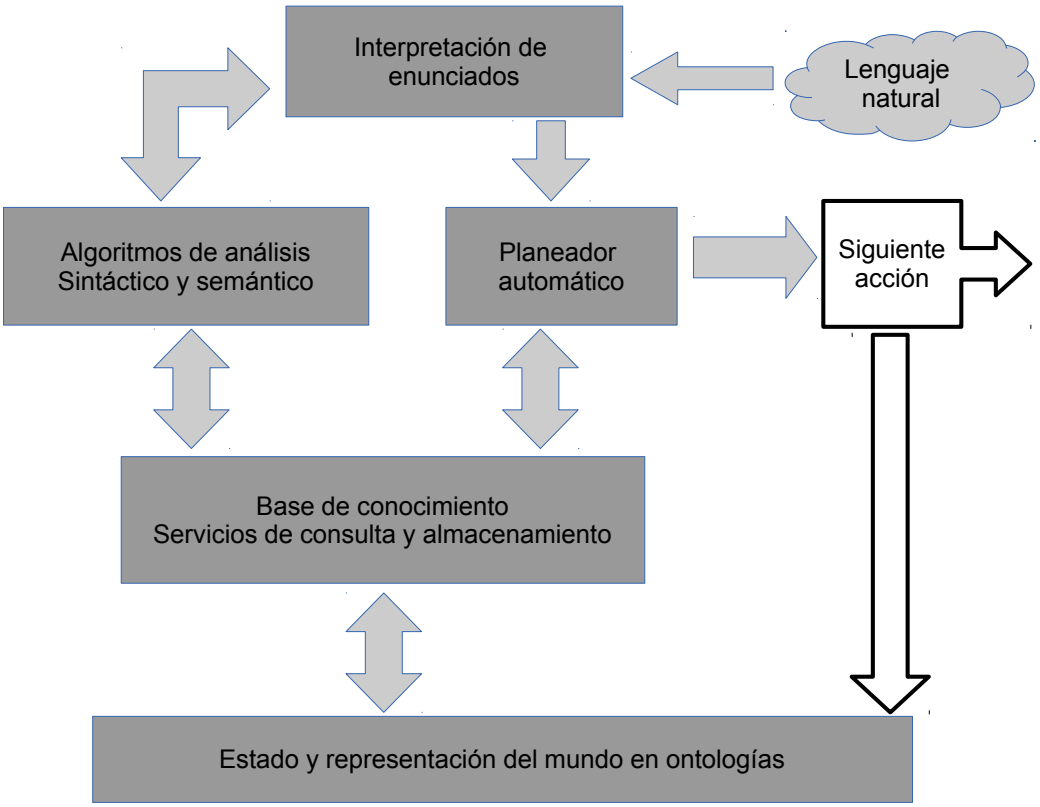


Figura 6.1 Diagrama general del software.

- Carga y almacenamiento de la ontología en archivos de texto.
- Recopilación de etiquetas de categoría gramatical para el vocabulario conocido.
- Consulta subclases, superclases y objetos de una categoría.
- Consulta del valor de un atributo usando la información expresada en la topología de los atributos y las propiedades algebraicas de las relaciones.

6.1.3 Intérprete de lenguaje

El intérprete de lenguaje es el responsable de etiquetar, separar y aterrizar las estructuras internas de la oración para asociarlos a un patrón de interpretación adecuado. Este software lo escribí en python. Desde el momento en que se recibe la oración en texto plano el software usan distintos algoritmos que incluyen:

- Sustitución de expresiones regulares, cambiar a singulares e identificar palabras compuestas.
- Etiquetado de categorías gramaticales con desambiguación léxica usando reglas.
- Identificación de constituyentes usando gramáticas CFG.
- Aterrizaje de frases nominales y preposicionales usando la base de conocimiento o generando expresiones para evaluación perezosa.
- Asociación de roles semánticos del enunciado de entrada a un patrón de interpretación.
- Generación de expresiones de significado y de manejo de la interacción verbal de respuesta del robot.

6.1.4 Planeador de acciones

El planeador de acciones lo escribí en lenguaje CLIPS como un sistema de reglas de producción. Este lenguaje ofrece evaluación de funciones aritméticas y lógicas con una sintaxis parecida a la de LISP. La principal característica del lenguaje CLIPS es ser un *expert shell* cuyo funcionamiento se basa en la descripción de hechos y el disparo de reglas dado un conjunto de precondiciones. CLIPS sigue el mismo paradigma de computación presentado anteriormente en el capítulo *Representación de conocimiento simbólico* en la subsección de *sistemas de reglas de producción*. Por lo tanto, el motor de inferencia mantiene la lista de hechos que disparan las reglas declaradas y busca satisfacerlas con los hechos presentes en cada momento. Cuando localiza una o más reglas que son candidatas a aplicarse, las agrega a una agenda y las ejecuta en orden de prioridad. Por defecto todas las reglas tienen la misma prioridad pero se puede definir manualmente en cada regla.

El paradigma de computación de CLIPS permite una traducción sencilla a partir del diseño de las redes jerárquicas de tareas a reglas de producción. Cada nodo de la gráfica corresponde una regla en CLIPS que debe expresarse en dos partes: antecedente y consecuente. El antecedente incluye la información

- **Solicitud de ejecución inmediata**. Corresponde a las expresiones generadas por el intérprete de lenguaje las cuales definen el tipo de actividad y sus parámetros aterrizados.
- **Precondiciones sobre el estado del mundo**. Corresponden a la descripción del estado del mundo que vuelve a la tarea relevante.
- **Nivel de prioridad de la regla**.

El consecuente de la regla incluye

- **Comando de ejecución directa o red de subtareas de sustitución**. Si la tarea es primitiva se muestra en consola el comando que se envía a módulos externos con los parámetros aterrizados. Si no es tarea primitiva se agregan los nodos que describen la red de sustitución, generando al vuelo identificadores únicos en cada solicitud de ejecución y estipulando las dependencias de orden entre ellas.
- **Efectos sobre el mundo**. Los efectos sobre el mundo incluyen la retracción y aseveración de hechos lo cual simula los efectos que deberían ingresar los módulos externos al hacer cambios en el mundo real.
- **Hechos de solicitud de desbloqueo**. Son reglas que administran la exploración del grafo de tareas.

Al poder especificar la prioridad de cada regla permite definir tareas que están latentes para su ejecución pero que por su baja prioridad no se ejecutan si siempre existe una tarea relevante con preferencia. Usando esta característica de manipulación de prioridad se propone especificar las reglas de ejecución de tareas en tres capas como se muestra en la figura 6.2

- **Alta prioridad**. Son reglas que infieren conocimiento y son del tipo *si dos cuartos están unidos mediante una puerta y la puerta está abierta, entonces los cuartos están conectados*
- **Prioridad estándar**. Son reglas que definen tareas que corresponden a un nodo en las redes jerárquicas siguen los lineamientos mencionados anteriormente.
- **Baja prioridad**. Son tareas que corresponden a un *método STN posiblemente aplicable*. Estos métodos se modelan como nodos que no difieren en estructura de cualquier otro, excepto que no tienen un orden de precedencia. Son partes desconectadas de la gráfica que pueden ejecutarse si las capas de prioridad superior no tienen un contexto relevante, lo que interrumpe el proceso normal de planeación.

Las expresiones de salida del intérprete de lenguaje siempre iniciarán la planeación a partir de un nodo en la capa de prioridad estándar. Los métodos de baja prioridad únicamente se ejecutan si no hay un contexto relevante para ejecutar un método en el proceso normal de planeación.

6.1.5 Arquitectura de software en el robot Justina

La forma en que distintos subsistemas se comunican en el robot Justina es mediante el software *blackboard* (programa homónimo al patrón arquitectónico que sigue), el cual

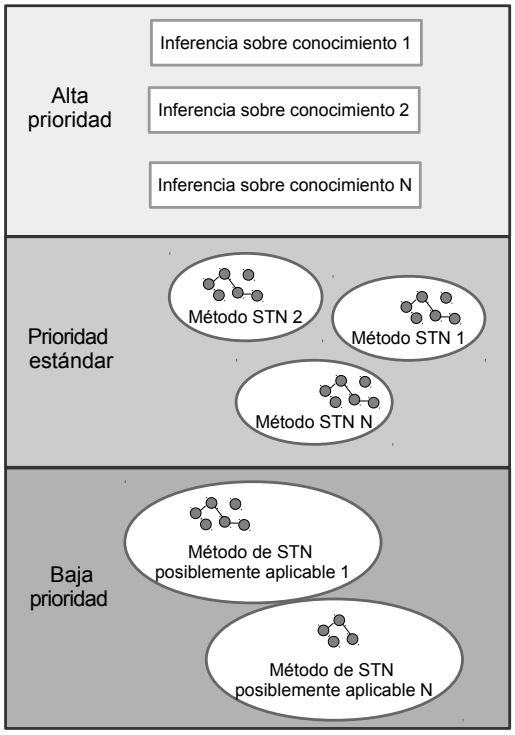


Figura 6.2 Diagrama general del software.

conecta un conjunto de programas independientes que trabajan cooperativamente sobre una estructura de datos común. Este programa es el almacén de datos central, donde leen y escriben los demás módulos coordinados por un componente de control. Así, el blackboard es un espacio común a los subsistemas para intercambiar mensajes y compartir variables en tiempo de ejecución. En la figura 6.3 se muestra un diagrama de esta arquitectura con los subsistemas que componen el estado actual del software en el robot *Justina*.

Para conectar un programa al blackboard se utilizan dos distintas *Interfaces de programación de Aplicaciones* (API) llamadas *Robotics* y *pyRobotics* para enlazar programas en C++ y python, respectivamente. Cada módulo enlazado define los comandos que puede ejecutar a petición de otro módulo, así como de las variables compartidas a las que está suscrito. Esto permitió encapsular el prototipo de esta tesis como un módulo llamado *LanguageUnderstandingPlanning*. Este módulo recibe una cadena de texto procedente del módulo de reconocimiento de voz y envía mensajes a los módulos motionPlanner, simpleTaskPlanner o speechGeneration, según sea el caso, para ejecutar acciones sobre el robot real [34].

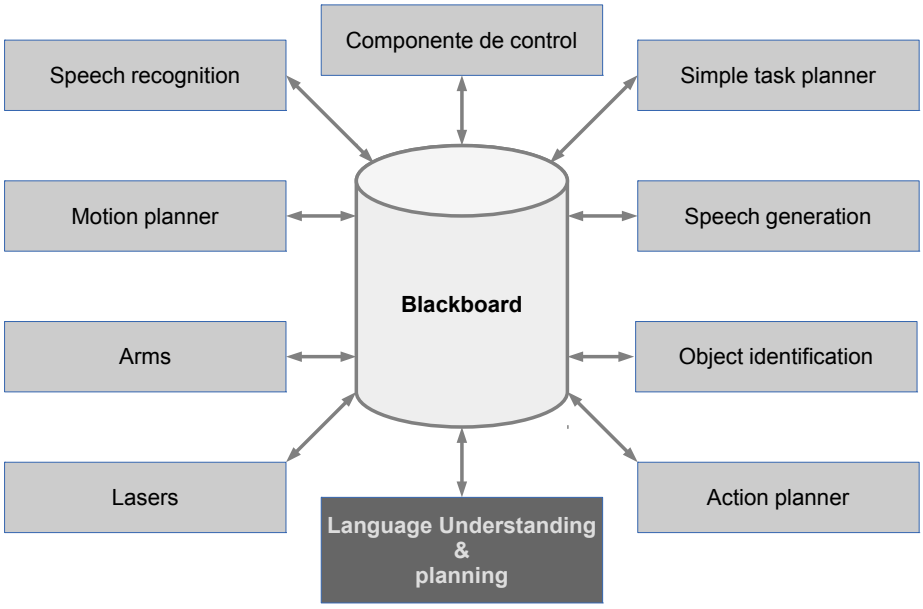


Figura 6.3 Arquitectura Blackboard.

6.2 Ejemplos de codificación de conocimiento

Se detallarán algunos ejemplos de cómo integrar conocimiento que un robot de servicio necesita tener para interpretar las oraciones en lenguaje planteadas anteriormente. El grado de detalle expresado en la ontología permanece siendo decisión del desarrollador de la aplicación y no existe una forma totalmente objetiva de saber el nivel de granularidad adecuado en la representación del mundo. En este trabajo se enfatiza el uso de la base de conocimiento como parte del léxico del interprete de lenguaje para resolver interacciones humano-robot de un robot humanoide de servicio doméstico.

6.2.1 Conocimiento en la ontología

El conocimiento espacial forma parte fundamental de conocimiento de sentido común y por lo tanto está ligado a la mayoría de otros campos de conocimiento que un humano o robot procesa. Las metáforas espaciales son muy comunes en la comunicación humana y se apoyan de conocimiento espacial previo para comunicar ideas que serían difíciles de comunicar sin él. Este tipo de conocimiento está aterrizado en la experiencia sensorial, es decir, el significado de los símbolos en la representación están definidos en términos de la experiencia de percepción y acción. El conocimiento espacial se expresa en distintas formas incluyendo procedimientos para llegar de un lugar a otro, mapas topológicos y

modelos geométricos del ambiente [29].

Kuipers en [29] realiza un esfuerzo por formalizar todos los niveles de percepción que un sistema debe tener. En ese trabajo se proponen niveles de distintos grados de abstracción, cada uno usando inferencia sobre el nivel más bajo. Así la jerarquía comprende *el nivel de control, nivel causal, nivel topológico y nivel métrico.*

En representaciones simbólicas dos tipos de conocimiento espacial resaltan por su utilidad: la topología del ambiente y la localización de objetos físicos.

Los mapas topológicos son una abstracción del ambiente descrito en un grafo, donde los nodos representan lugares y los vértices representan conexiones o caminos físicos. Debido a que son una representación puramente simbólica, los mapas topológicos permiten razonamiento de alto nivel a diferencia de los mapas métricos.

Por otro lado, la representación simbólica de la localización de objetos se lleva a cabo al relacionar un objeto con otro objeto o lugar mediante una relación apropiada. Estas relaciones pueden ser absolutas o relativas. Por ejemplo, una relación como *está_sobre* es absoluta porque su validez no depende de la naturaleza de los objetos o lugares involucrados. Por el contrario, una relación como *está_junto* es relativa porque depende del tamaño relativo de los objetos o lugares involucrados. Dos casas separadas dos metros se consideran estar una junto a otra pero dos hormigas separaras dos metros no se considerarían contiguas [24].

Considere que se quiere representar una escena con dos cuadrados, un circulo y un triángulo como se muestra en la figura 6.4. Se deseas codificar las posiciones respecto al piso y respecto a otras figuras.

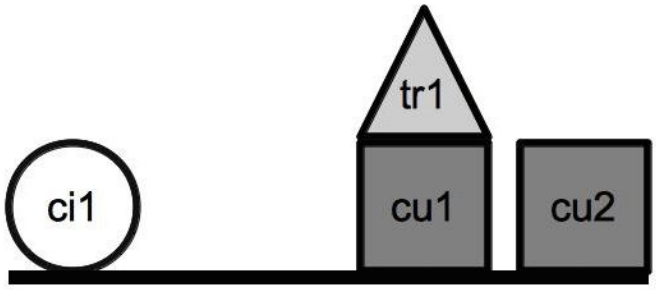


Figura 6.4 Escena de relaciones espaciales.

Se empieza con la ontología vacía excepto por los nodos obligatorios anteriormente mencionados

is_kind_of(stuff, top)

is_kind_of(attribute, top).

Se crea una categoría de atributos llamada *location*, de ella podemos derivar todas las categorías de atributos necesarias. Para este ejemplo consideraremos las relaciones *on, right_to, left_to, near, far.*

is_kind_of(location, attribute)

is_object_of(on, location)

is_object_of(right_to, location)

is_object_of(left_to, location)

is_object_of(near, location)

is_object_of(far, location)

Estas relaciones tienen distintas propiedades en la representación del mundo que se desea formar. Por ejemplo, si se sabe que *ci1 está encima del piso* no podemos concluir que *el piso está encima de ci1.* No obstante, si *cu1 está cerca de cu2* sí podemos concluir que *cu2 está cerca de ci2.* Por tanto, es útil diferenciar relaciones simétricas y asimétricas. Asimismo, si *ci1 está a la izquierda de cu1 y cu1 está a la izquierda de cu2* se quisiera establecer que *ci1 está a la izquierda de cu2*, es decir que es una relación transitiva. Por lo tanto se aumentan los nodos:

attribute_property(on, transitive)

attribute_property(right_to, transitive)

attribute_property(left_to, transitive)

attribute_property(near, symmetric)

attribute_property(far, symmetric)

finalmente podemos crear una jerarquía de figuras geométricas:

is_kind_of(figure, stuff)

is_kind_of(triangule, figure)

is_kind_of(square, figure)

is_kind_of(circle, figure)

is_object_of(tr1, triangule)

is_object_of(cu1, square)

is_object_of(cu2, square)

is_object_of(ci1, circle)

y codificar las relaciones espaciales entre ellos:

far(ci1, cu1)

far(ci1, cu2)

near(cu1,cu2)

on(tr1,cu1)

left_to(ci1, cu1)

left_to(ci1, cu2)

right_to(ci2, cu1)

right_to(cu1, ci1)

Se podrían agregar nuevas atributos y valores para aumentar el grado de detalle de la representación como el color de los objetos, peso, etc. Los siguientes nodos representan la información implícita congruente con los hechos anteriores:

far(cu1, ci1)

far(cu2, ci1)

near(cu2,cu1)

left_to(ci1, cu2)

right_to(cu2, ci1)

Lo anterior concuerda con lo que se quería modelar para este micromundo. Aplicar propiedades sobre las relaciones permite codificar cualquier tipo de relaciones en otros micromundos que siguen los mismos lineamientos algebraicos. Por ejemplo, la relación *casado_con* es simétrica, *es_jefe_de* es transitiva y *conectado_por_pasillo* es transitiva y simétrica.

La suposición de un mundo abierto en contraste con un mundo cerrado se refiere a la decisión de diseño que se debe tomar en torno a la completitud del conocimiento en el dominio. En un mundo cerrado, cualquier hecho que no se pueda probar como verdadero es considerado falso, mientras que en un mundo abierto puede considerarse como verdadero, falso o desconocido. Asumir un mundo cerrado facilita negar un hecho al retirarlo de la base de conocimiento. Si el robot toma una botella de la mesa es suficiente con eliminar el hecho *la botella está sobre la mesa* y agregar *el robot sostiene la botella* para mantener la integridad de la información [24].

Como se ha dicho con anterioridad, los símbolos que se declaran en la ontología cumplen con el propósito de ser parte del diccionario o *léxico* de las rutinas de comprensión de lenguaje natural. Por lo tanto, el ejemplo se seguirá comentando en idioma inglés.

Además, se considera un modelo del mundo similar al requerido en las competencias *robocup@home* y *rockIn@home*.

Primero se propone subdividir la categoría de *stuff* en *person, item y place*. A su vez *person* puede subdividirse en *woman y man* y cada una de sus instancias se representarían con vértices como *is_object_of(mary, woman)* o *is_object_of(john, man)*. Por otro lado, *item* puede subdividirse en *furniture, food, tool, chemical_products* y cada una de estas categorías puede tener muchos refinamientos que se consideren pertinentes. De la categoría *item* se refinan instancias y categorías de *chair, table, shelf, drink, candy, chemical_product* y demás objetos involucrados en actividades domésticas. La clase place se refina en *zone, room, house y city*. Si se considera pertinente, algunas categorías como *table y shelf* puede ser consideradas tanto un refinamiento de *item* como de *place*.

Otro aspecto de importancia en el modelado del micromundo son los atributos y sus propiedades. Para este ejemplo se propone tener como refinamiento de *attribute* las categorías *physical_status, funcitional_status, social_status y location*. La categoría de physical_status debe contener refinamientos como *color, shape, size, weight, temperature, age*. Cada una de estas categorías tiene como instancias a palabras como *purple, round, small, light, cold y old*, respectivamente. Por otro lado, la categoría de funcional_status contiene refinamientos como *ready, graspable, accesible, conected_to*. La categoría de social_status debe contener refinamientos como *married, widow, friends_with*, etc.

6.2.2 Diseño de patrones de interpretación

Los tipos de declaraciones codificadas en el sistema son:

- **Declaración de una nueva categoría de objetos o atributos**. Esto permite aumentar el vocabulario de la ontología ingresando oraciones como *candy is a kind of food* y *weight is a sort of physical attribute*.
- Declaración de una instancia de una clase de objeto o atributo. Incluye enunciados como *red is an adjetive of color* y *chair_2 is an instance of chair*
- Declaración del valor de un atributo de un objeto o categoría. Esto permite relacionar un valor y un objeto mediante un atributo como en *the tall man is single* y en *all drinks are on kitchen table*

En el cuadro 6.1 se presenta un patrón para interpretar el tipo de declaraciones del tercer punto. Estas oraciones contienen una frase nominal, el verbo *is* y un valor de atributo al cual se quiere relacionar con el objeto al que hace referencia la frase nominal.

Dependencia conceptual	(MBUILD (ACTOR: robot) (OBJECT: classobject ATTRIBUTE: =kb.get_parent(value) VALUE: value))			
Confirmación verbal	(Action SAY mesage " Ok, classobject is kb.get_parent(value) value, right? ")			
Comando al planeador +	(Action SAY mesage =kb.add_fact(classobject, kb.get_parent(value), value))			
Comando al planeador -	(Action SAY mesage "Please rephrase the sentence")			
Parámetros	**Constituyente**	**Tipos semánticos**	**Palabras clave**	**Valor en ausencia de especificación**
action	verb	.	is, are	-
classobject	noun_phrase	stuff	-	-
value	adjetive	attribute	-	-

Cuadro 6.1 Patrón de interpretación de declaración de un valor de atributo en una clase u objeto.

Como se observa en la expresión de dependencia conceptual del cuadro 6.1, el patrón usa una función de la base de conocimiento para ingresar los nodos correspondientes a la ontología para actualizar el modelo del mundo. Además, para inferir la relación que se debe asignar entre el adjetivo y el objeto se consulta el padre del adjetivo encontrado en la oración. Por ejemplo, si se ingresa el enunciado *"the man in the livigroom is tall"* el sistema asociaría $action = is$, $classobject = jack$ y $value = tall$. Al sustituir estos valores aterrizados en las expresiones del patrón de interpretación se evalúa las funciones $kb.get\_parent(tall)$ y $kb.add\_fact(classobject, kb.get\_parent(value), value))$ resultando en:

Dependencia conceptual: *MBUILD (ACTOR: robot) (OBJECT: sam ATTRIBUTE: shape VALUE: tall))*

Confirmación verbal: *(Action SAY mesage "Ok sam is shape tall, right?")*

Comando al planeador +: *(Action SAY mesage "Ok")* y se agregan nuevos nodos a la ontología.

Comando al planeador -: *(Action SAY mesage "Please rephrase the sentence")*

Otros patrones pueden usar funciones similares para especificar el manejo de preguntas cuya respuesta se genera al consultar la base de conocimiento. En este trabajo se consideran dos clases de preguntas:

- **Consultar el valor de un atributo que califica un objeto o clase**. Esto incluye preguntas como *"what is the temperature of the livigroom?"* o *"what is the color of the bottles in the fridge?"*. Además se incluye el caso especial cuando se solicita la ubicación usado el adverbio *where* como en *"where is John?"*
- **Consulta de objetos que cumplen con un conjunto de atributos**. Como

"Who is married?" o *"what rooms are connected to bedroom_1?"*

En el cuadro 6.2 se muestra el seudocódigo de uno de los patrones de interpretación para preguntas del primer tipo de preguntas.

Dependencia conceptual	(MBUILD (ACTOR: speaker) (OBJECT: classobject ATTRIBUTE: attribute VALUE: =kb.consult(classobject, attribute)))			
Confirmación verbal	(Action SAY mesage "Ok you ask for what is the attribute of classobject, right?")			
Comando al planeador +	(Action SAY mesage =kb.consult(classobject, attribute))			
Comando al planeador -	(Action SAY mesage "please rephrase the sentence")			
Parámetros	**Constituyente**	**Tipos semánticos**	**Palabras clave**	**Valor por defecto**
wh_adverb	adverb	-	what, which	-
action	verb	.	is, are	-
classobject	frase nominal	stuff	-	-
attribute	frase nominal	attribute	-	-

Cuadro 6.2 Patrón de interpretación de preguntas de consulta a un atributo de una clase u objeto.

6.2.3 Ejemplo de diseño de tareas no primitivas

Considere la tarea no primitiva de *fetch*. Se propone la descomposición de nodos como se muestra en las figuras 6.5, 6.6 y 6.7. Las imágenes muestran la topología de la actividad. Como se observa, este modelo en particular propone a fetch como una tarea no primitiva totalmente ordenada, lo que implica que la tarea se resuelve con la secuencia de tareas primitivas *go, recognize, approach, grasp, go, recognize, aproach y hand_over* en ese orden, cuyos parámetros se van sustituyendo automáticamente dependiendo de la invocación inicial al primer nodo.

Como se observa en la figura 6.5, la descomposición de esta tarea se describe con tres tareas. La primera tarea a sustituir es *go_aproach* la cual recibe como único parámetro el objecto al que debe acercarse. Cuando esta tarea termine de ejecutarse, se sustituye la tarea primitiva *grasp* especificando el objeto que debe agarrar. Finalmente, se ejecuta la tarea *deliver_cargo*, con la termina el comportamiento. Cada una de estas tareas deben describirse mediante sus parámetros de entrada, precondiciones y efectos. Por ejemplo el nodo *go_aproach* se describe como:

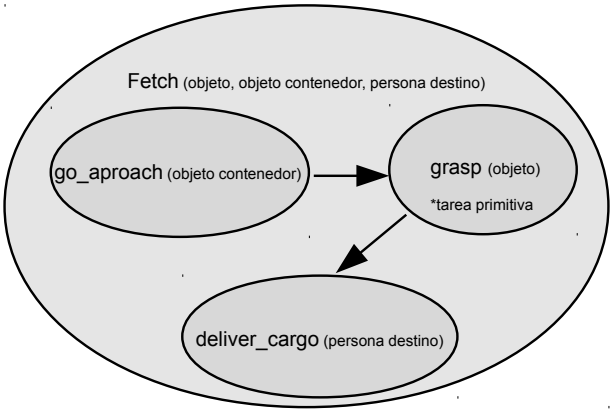


Figura 6.5 Composición de redes de tareas para fetch.

Identificador	go_aproach
Parámetros de entrada	objeto
Precondiciones	-
Efectos +	-
Efectos -	-
Grafo de sustitución	V = {go(consult(location, objeto)), recognize(objeto), aproach(objeto)}, E = {(go, recognize), (recognize, aproach)}

Cuadro 6.3 Descripción del nodo go_aproach en la red jerárquica de tareas de fetch.

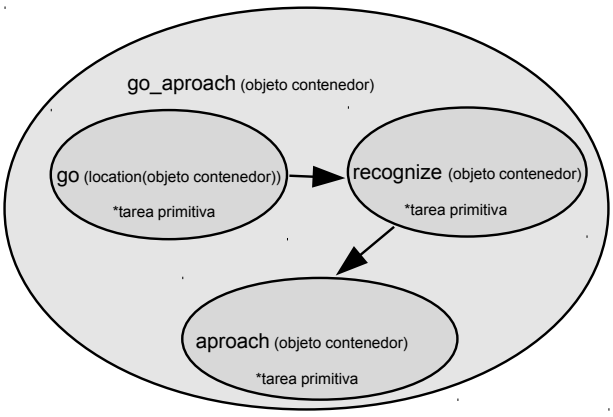


Figura 6.6 Composición de redes de tareas para go_aproach.

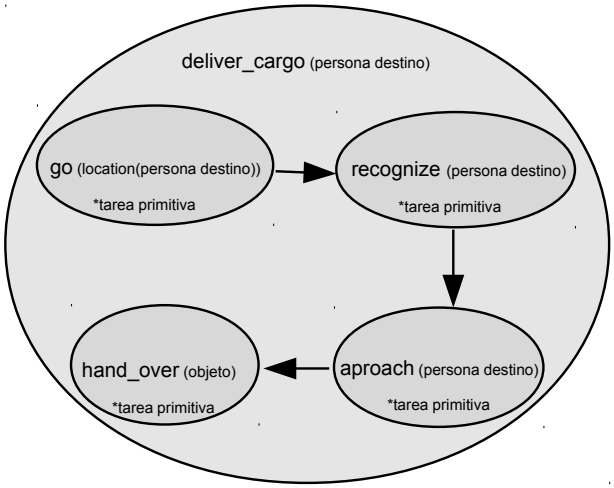


Figura 6.7 Composición de redes de tareas para deliver_cargo.

Capítulo 7

Evaluación

El sistema únicamente puede realizar tareas e interpretaciones que tiene codificadas, lo cual se captura en distintas estructuras como la representación del conocimiento interna al robot. Dependiendo de las capacidades del robot y al lenguaje natural controlado que se desea aceptar, la configuración del sistema debe cambiar. El funcionamiento del sistema en una aplicación depende de la correcta codificación del micromundo.

La forma en que sistemas similares se han evaluado incluye conducir pruebas con un robot real en situaciones controladas [42] y reportar el tipo de diálogos que el sistema puede entablar [43], [49]. Los trabajos planteados como un lenguaje natural controlado reportan análisis de expresividad sobre el conocimiento que codifica el lenguaje y muestran ejemplos de asignación del significado formal en varios casos típicos en el contexto de la aplicación [44]. Competencias como *Robocup@home* y *RockIn@home* proveen lineamientos más específicos sobre el comportamiento esperado del robot y existen referencias que toman un subconjunto de las pruebas en competencia para evaluar al sistema [46].

Para reportar pruebas cuantitativas se utilizan corpus anotados que sirven como *gold standard* para establecer una comparación directa de rendimiento a un sistema ideal. Incluso existen unos pocos corpus anotados específicamente para ordenar robots en lenguaje natural. No obstante, la mayoría de los sistemas reportados se entrenan mediante algoritmos de aprendizaje estadísticos basados en ejemplos, en vez de interpretaciones manualmente codificadas [47], [48], [49].

7.1 Evaluación de comprensión de lenguaje

Para evaluar el rendimiento del sistema en la interpretación de lenguaje se propone dar descripción de los lenguajes naturales controlados que se pueden definir para

ser aceptados por el sistema y sobre los lenguajes formales que puede producir. Después se muestra un fragmento de un diálogo que el sistema logra entablar al tener codificado el micromundo de un robot de servicio doméstico. Después se muestran los resultados de adaptar el sistema para la prueba *Speech understanding* de la competencia *RockIn@home* edición 2014, la cual consiste en la separación e identificación de roles semánticos en comandos en lenguaje natural a un robot doméstico de servicio. Finalmente, se reporta el rendimiento del sistema reconfigurado para procesar el corpus SEMEVAL 2014 que se compone de comandos espaciales a un robot hipotético que manipula figuras de colores en un espacio tridimensional.

7.1.1 Descripción de los lenguajes aceptado y generado

El lenguaje natural controlado que acepta el sistema depende en su nivel superior de los patrones de interpretación, los cuales son una lista de parámetros que se deben aterrizar para producir una expresión de interpretación. Cada parámetro puede ocurrir en cualquier orden en la oración. Por lo tanto, cada patrón acepta el conjunto de enunciados que contenga una permutación de los parámetros descritos $((\sigma(Parametro1), ..., \sigma(ParametroN))$. El lenguaje aceptado L_{in} es la unión del lenguaje interpretado por cada uno de los patrones:

$$L_{in} = S \mid (((\sigma(Parametro1\_Patron1), ..., \sigma(ParametroN1_{P}atron1)) \cup ...$$
$$\cup((\sigma(Parametro1\_PatronM), ..., \sigma(ParametroNM_{P}atronM))) \subset S_{param}$$

donde S es un enunciado en lenguaje natural, S_{param} es el conjunto de roles semánticos presentes en S, M es el número de patrones de interpretación en el sistema y $N1, N2, ..., NM$ son el número de parámetros en cada uno de los M patrones.

Cada uno de esos parámetros se describe sintácticamente por su constituyente o categoría gramatical $Constituent_i$ y por palabras clave $Keywords_i$ que deba contener. Por lo tanto, la asociación de parámetros es función del conjunto Pos de etiquetas de categorías gramaticales y las gramáticas para separar frases nominales G_{np} y preposicionales G_{pp}, las cuales aceptan los lenguajes L_{np} y L_{pp}, respectivamente. De esta forma, el conjunto de palabras $Words\_param_i$ con etiquetas de categorías gramaticales Pos_param_i se asocia a un parámetro descritos en el patrón si:

$$\begin{aligned}
Pos_param_i \in L_{np}, \quad & si \quad Constituent_i = noun_phrase \\
Pos_param_i \in L_{pp}, \quad & si \quad Constituent_i = prepositional_phrase \\
Pos_param_i = Constituent_i, \quad & si \quad Constituent_i \in Pos \\
Keywords_i \in Words_param_i &
\end{aligned}$$

Además, las frases nominales y preposicionales se aterrizan a símbolos en la ontología o a una expresión de *evaluación perezosa*, si el parámetro se aterriza a un nodo de la ontología, puede restringuirse según el tipo semántico del objeto aterrizado. El objeto aterrizado $Grounded\_param\_i$ producto de aplicar el algoritmo de aterrizaje de frases nominales a un conjunto de palabras $Words\_param_i$ debe cumplir con ser el tipo semántico $semantic\_type_i$ según el estado de la ontología O en la base de conocimiento $KB(O)$. Entonces debe cumplirse que:

$$KB(O) \models is\_kind\_of(Grounded\_param\_i, semantic\_type_i)$$

Cuando la asociación de roles semánticos es exitosa, cada uno de ellos tiene asociado un símbolo que puede ser un nodo de la ontología o una expresión para su evaluación posterior, entonces cada parámetro tiene un símbolo asociado de la forma $Parametro1 = objeto\_aterrizado\_1, ..., ParametroN = objeto\_aterrizado\_N$. El símbolo o expresión aterrizada de cada parámetro se usa como parámetro de la función que genera las expresiones de significado y órdenes al planeador. Los lenguajes L_{out} son el conjunto de expresiones que genera una función que recibe como parámetros un subconjunto de los símbolos aterrizados:

$$L_{out} = f(O, objeto\_aterrizado\_1, ..., objeto\_aterrizado\_N)$$

La función f que genera la expresión de salida puede variar según la especificación del lenguaje que se desee producir, ya sea dependencias conceptuales, comandos RCL (corpus SEMEVAL), comandos CFR (competencia RockIn@home), comandos al planeador presentado en esta tesis u otro lenguaje que requiera la aplicación. La función f puede variar desde una simple concatenación de cadenas hasta generación de lenguaje natural.

7.1.2 Pruebas con la prueba Speech understanding de la competencia RockIn

La prueba *Speech Understanding* de la competencia de robótica *RockIn@home* intenta establecer un esquema estándar para evaluar la habilidad del robot para procesar adecuadamente comandos por voz en un contexto doméstico. La prueba consiste en recibir oraciones mediante señales de voz, reconocer las palabras pronunciadas y generar una expresión formal de significado que sigue una estructura de comando/argumentos de acuerdo al formato *Command Frame Representation (CFR)*, estipulado por el comité de la competencia. El archivo de audio .wav de la señal de voz, su transcripción y el comando CFR se almacenan en una memoria USB, para su posterior revisión por parte de los jueces de la competencia [41].

Por el planteamiento de la prueba, el sistema hace uso de la cadena de texto que produce el reconocedor de voz para generar la expresión en formato CFR. El sistema no genera un plan de acción ya que sale del alcance de esta prueba en específico. En el cuadro 7.1 se muestran algunos ejemplos de posibles oraciones enunciadas y su respectiva representación en CFR.

Enunciado de entrada	**Comando CFR**
"bring the bottle to the couch"	BRINGING(theme: the bottle, goal: to the couch)
"go close to the shower"	MOTION(goal: close to the shower)
"give me one apple from the table"	GIVING(recipient: me, theme: one apple from the table)
"please pick my mobile phone and put it on the chair near the table"	TAKING(theme: my mobile phone)#PLACING(theme: it, goal: on the chair near the table)

Cuadro 7.1 Enunciados ejemplo de la prueba *Speech Understanding* y su representación en el lenguaje CFR.

La prueba tiene un planteamiento basado fuertemente en la sintaxis de las oraciones y para generar el lenguaje de salida es suficiente separar los roles semánticos como *the bottle* y *on the chair near the table*, pero no se requiere aterrizarlos a una representación del mundo particular. Al reconfigurar el sistema para esta prueba, modifiqué las ontologías para incluir el vocabulario estipulado, ajusté las gramáticas y modifiqué el algoritmo de aterrizaje de roles semánticos. Usando como guía los enunciados que se presentaron como muestra por el comité de la competencia *RockIn@home*, codifiqué un total de 17 patrones de interpretación. En la figura 7.1 se muestran algunos de los patrones usados. El patrón *A* de la figura 7.1 interpreta enunciados como *go to the livingroom, move to the table in bedroom* y *move near a person*. El patrón *B* se diseñó para enunciados como *take a drink from the bridge, grasp my glasses on my bed* y *take a pecil on the table*. El patrón *C* procesa enunciados como *get me a coke from the fridge to the couch*.

A

Roles semánticos	Palabras clave	Constituyente	Tipo semántico
what_action	go, move, enter	verb	-
what_goal	to, over, near	prepositional_phare	place, item
Expresión de salida	MOTION(goal:'-*what_goal*-')		

B

Roles semánticos	Palabras clave	Constituyente	Tipo semántico
what_action	take, grasp, grab	verb	-
what_object	-	noun_phare	item
what_source	in, on, from	prepositional_phrase	item, place
Expresión de salida	TAKING(theme:'-*what_object*-', source:'-*what_source*-')		

C

Roles semánticos	Palabras clave	Constituyente	Tipo semántico
what_action	bring, get, fetch	verb	-
what_object	-	noun_phrase	item
what_source	in, on, from	prepositional_phrase	item, place
what_goal	to	prepositional_phrase	item, place
what_beneficiary	-	noun_phrase	person
Expresión de salida	BRINGING(theme:'-what_object-', source:'-what_source-', goal:'-what_goal-', beneficiary:'-what_beneficiary-')		

Figura 7.1 Ejemplos de patrones de interpretación para la competencia RockIn prueba speech understanding.

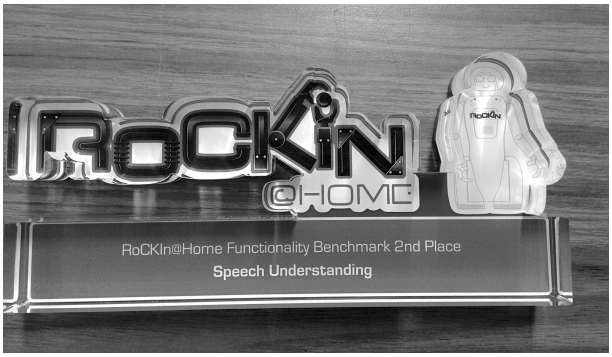


Figura 7.2 Trofeo de segundo lugar en la prueba *Speech Understanding* en la competencia *RockIn@home* 2014.

El equipo Pumas ganó el segundo lugar en la prueba Speech Understanding. En la figura 7.2 se muestra una foto del trofeo.

7.1.3 Pruebas con el corpus SEMEVAL 2014

Para contrastar el sistema propuesto en esta tesis contra otros métodos de procesamiento de lenguaje natural, presento los resultados obtenidos con un corpus anotado de un dominio similar a un robot doméstico. El corpus se publicó por un grupo de inteligencia artificial de la universidad de Leeds en el Reino Unido proponiendo la tarea compartida *Supervised Semantic Parsing of Robotic Spatial Commands*. El corpus de esta tarea provee anotaciones semánticas de 3,409 comandos espaciales (41,158 palabras) para manipular figuras geométricas de distintos colores en un espacio tridimensional. Los enunciados provienen del uso de un juego en línea diseñado para capturar distintas escenas en un tablero tridimensional con ambigüedad espacial, confusión de color, marcas en la escena y grupos de figuras grandes y pequeños. Los usuarios recibieron dos escenas distintas y escribieron órdenes a un robot hipotético para lograr que ambas escenas sean iguales. El treebank mapea las oraciones a un lenguaje formal llamado Robot Control Language RCL. El corpus se divide en tres partes. Los primeros 500 enunciados del corpus corresponden a una muestra de ejemplo. De la oración 501 a la 2500 corresponden a la sección de entrenamiento. Las oraciones 2501 a la 3409 corresponden al conjunto de prueba [50].

Enunciado de entrada	**Comando RCL**
place green pyramid on top of red brick	(event: (action: move) (entity: (color: green) (type: prism)) (destination: (spatial-relation: (relation: above) (entity: (color: red) (type: cube)))))
place the red pyramid sitting on top of the red brick on top of the yellow one	(event: (action: move) (entity: (color: red) (type: prism) (spatial-relation: (relation: above) (entity: (id: 1) (color: red) (type: cube)))) (destination: (spatial-relation: (relation: above) (entity: (color: yellow) (type: type-reference) (reference-id: 1)))))
pick the red block and put it above the purple block	(sequence: (event: (action: take) (entity: (id: 1) (color: red) (type: cube))) (event: (action: drop) (entity: (type: reference) (reference-id: 1)) (destination: (spatial-relation: (relation: above) (entity: (color: magenta) (type: cube))))))

Cuadro 7.2 Ejemplos de enunciados del corpus SEMEVAL 2014 tarea 6 con significado en lenguaje RCL.

En el cuadro 7.2 se muestran algunos ejemplos de enunciados en lenguaje natural con su respectivo comando en lenguaje RCL. Como se observa, en el lenguaje RCL no se aterrizan las frases nominales a instancias de objetos en la representación del mundo. En el lenguaje RCL, se generan expresiones que asocian la descripción del objeto del rol semántico con un conjunto de expresiones *(campo: valor)*. Para generar estas descripciones modifiqué el algoritmo de aterrizaje de frases nominales con el fin que las expresiones de *evaluación perezosa* coincidieran con lo estipulado por el lenguaje RCL.

Por la naturaleza manual de la reconfiguración del sistema para procesar el corpus, no utilicé completamente la sección de entrenamiento. En vez, consulté el conjunto de prueba correspondiente a los primeros 500 enunciados para diseñar la ontología y

los patrones de interpretación. Para generar el lenguaje RCL, la ontología tuvo un total de 76 nodos para modelar el vocabulario, ajusté la separación y aterrizaje de frases nominales, y codifiqué un total de 11 patrones de interpretación. En la figura 7.3 se muestra el rendimiento del sistema para generar las expresiones RCL así como la aportación que hace cada patrón al total de oraciones interpretadas.

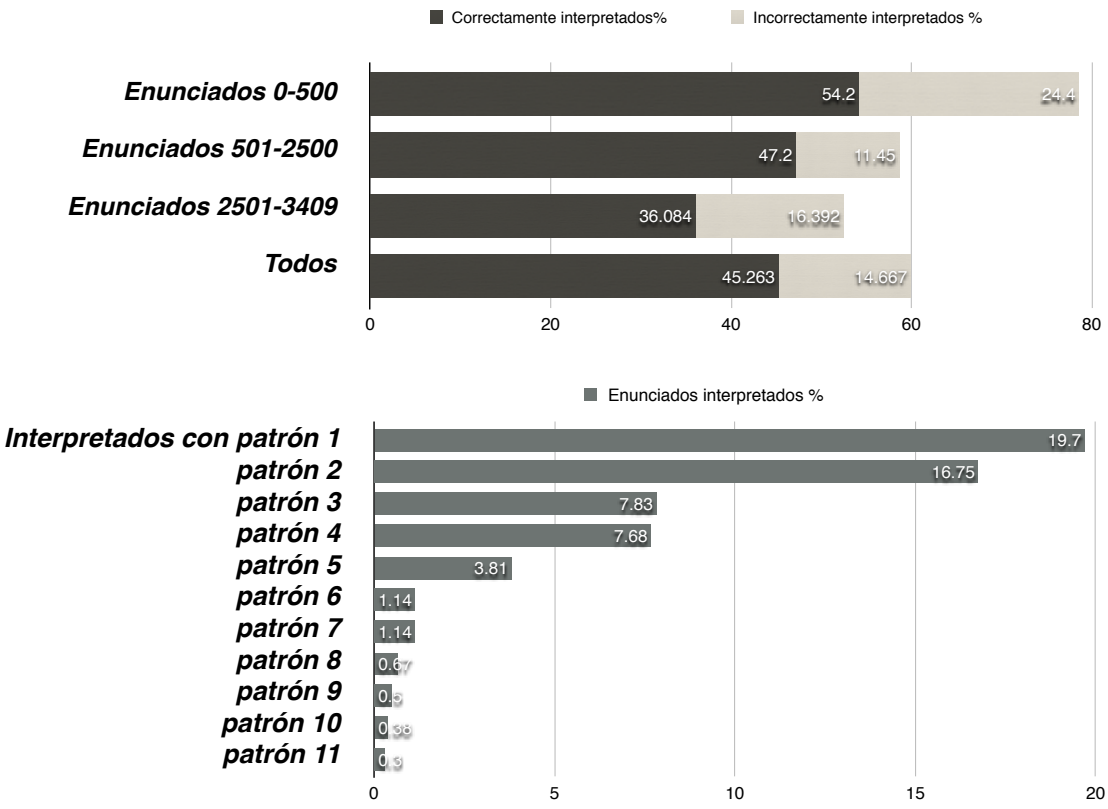


Figura 7.3 Rendimiento al procesar el corpus SEMEVAL 2014.

Como se observa en la figura 7.3, el sistema procesa correctamente el 54 % de los primeros 500 enunciados los cuales se revisaron a mano para reconfigurar el sistema. Al procesar la sección de prueba del corpus (de la oración 2501 a la 3409) el rendimiento cae hasta un 36 %. Del total de las oraciones en el corpus, casi el 60 % produce una expresión RCL de las cuales el 75 % de ellas es exactamente igual a las expresiones del corpus anotado. Del total de enunciados interpretados, el 90 % se interpretan usando los cinco patrones más frecuentes.

Las métricas de rendimiento en los conjuntos de oraciones de la 500 a la 2500 y de la 2500 a la 3409 son:

$$Precision_{500,2500} = 0.8 \quad Exhaustividad_{500,2500} = 0.47 \quad Valor\_F_{500,2500} = 0.30$$
$$Precision_{2500,3409} = 0.69 \quad Exhaustividad_{2500,3409} = 0.36 \quad Valor\_F_{2500,3409} = 0.24$$

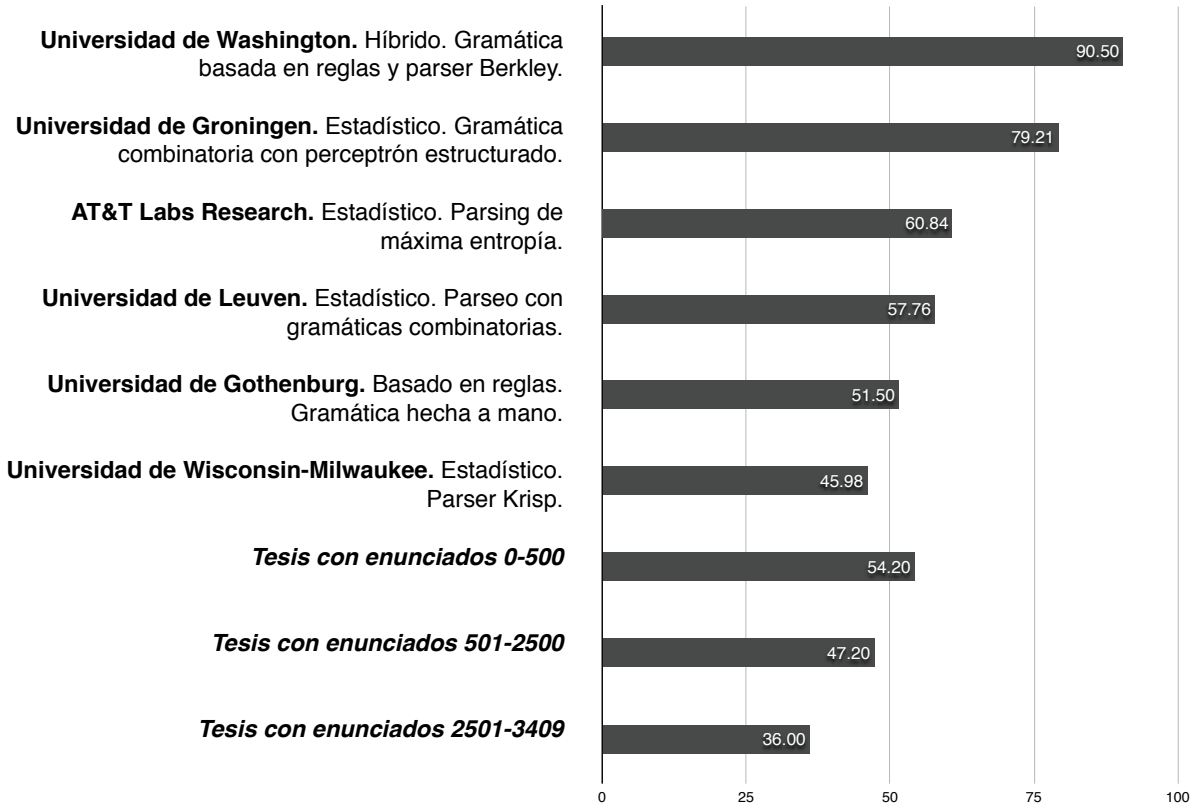


Figura 7.4 Comparación con resultados reportados en el SEMEVAL 2014 tarea 6, % de exhaustividad.

En el figura 7.4 se muestran los resultados encontrados y se comparan contra los resultados oficiales reportados. Aunque los resultados presentados corresponden al procesamiento de los mismos enunciados, los resultados encontrados en esta tesis no son comparables con los resultados oficiales debido a que no se entrenaron con el mismo conjunto de oraciones. Los resultados oficiales se obtuvieron al entrenar los sistemas con los enunciados del 501 al 2500, mientras que en este trabajo se entrenó con los enunciados del 1 al 500. Los resultados muestran que el sistema tiene un rendimiento inferior a otros. En general, se reportan mejores resultados mezclando gramáticas para el inglés con parsers estadísticos.

Con los patrones usados para esta prueba, el sistema comete dos tipos de errores: genera expresiones que no concuerdan con lo estipulado en el *gold standard* o no genera ninguna expresión. En el cuadro 7.3 se muestran algunos enunciados malinterpretados por el sistema, donde las letras marcadas en negritas destacan las diferencias en las expresiones.

Enunciado de entrada	**Anotación del corpus RCL**	**Expresión generada por el sistema**
place blue block on top of single red block	(event: (action: **drop**) (entity: (color: blue) (type: cube)) (destination: (spatial-relation: (relation: above) (entity: (indicator: individual) (color: red) (type: cube)))))	(event: (action: **move**) (entity: (color: blue) (type: cube)) (destination: (spatial-relation: (relation: above) (entity: (indicator: individual) (color: red) (type: cube)))))
pick up the grey triangle and place it on top of the red cube which is placed on a yellow cube	(sequence: (event: (action: take) (entity: (id: 1) (color: grey) (type: prism))) (event: (action: drop) (entity: (type: reference) (reference id: 1)) (destination: (spatial relation: (relation: above) (entity: (color: red) (type: cube) **(spatial relation: (relation: above) (entity: (color:yellow) (type: cube))**)))))))	(sequence: (event: (action: take) (entity: (id: 1) (color: grey) (type: prism))) (event: (action: drop) (entity: (type: reference) (reference id: 1)) (destination: (spatial relation: (relation: above) (entity: (color: red) (type: cube))))))
pick the which block on top of the sky blue block and place it over the red block which is near to bunch of blue blocks	**(sequence: (event: (action: take) (entity: (id: 1) (type: cube) (spatial relation: (relation: above) (entity: (color: cyan) (type: cube))))) (event: (action: drop) (entity: (type: reference) (reference id: 1)) (destination: (spatial relation: (relation: above) (entity: (color: red) (type: cube) (spatial relation: (relation: nearest) (entity: (color: blue) (type: cube group))))))))**	-

Cuadro 7.3 Ejemplos de enunciados malinterpretados del corpus SEMEVAL 2014 tarea 6.

La primera fila del cuadro 7.3 presenta un enunciado donde no se tiene consenso sobre el sinónimo de una palabra. En algunos casos la anotación del corpus usa algún sinónimo y en otros caso otro. En este ejemplo, *place* se sustituyó por *move* pero el corpus muestra *drop*. La segunda fila del cuadro presenta un ejemplo cuando el sistema interpreta únicamente un fragmento del enunciado y le asigna un significado incompleto. En ese ejemplo el sistema falla en agregar la última referencia espacial *"which is placed on a yellow cube"*. El enunciado de la última fila es ejemplo de cuando, por la presencia de errores gramaticales y la complejidad de las referencias espaciales, ningún patrón del sistema logra interpretarlo.

7.1.4 Pruebas implementando una interfaz de lenguaje natural para controlar un robot doméstico

Los patrones de interpretación permiten estipular qué acción enviar enseguida al planeador de tareas, esto permite que desde el patrón de interpretación se puedan diseñar interacciones verbales con el usuario.

Para probar esta capacidad del sistema, diseñé un total de 23 patrones y una ontología con 250 nodos con el fin de que el sistema pueda mantener una conversación simple, donde el usuario describe y pregunta sobre la disposición de objetos en un departamento. La ontología comienza con la jerarquía de categorías objetos, lugares, personas y atributos comunes en la descripción de este tipo de micromundos. No obstante, los objetos no tienen propiedades asignadas y es durante el curso de la conversación que nuevo conocimiento se integra para usarse en el aterrizaje de los parámetros de la interpretación de las oraciones.

	Parámetros aterrizados		**Comando al planeador**	**Información agregada o eliminada**
USUARIO: ***marie is in the office***	**Object:** **Action:** **Location:**	marie = marie is = is in the office = the_office	(ACTION: say_listen MSG: "ok, location of marie is the_office"	+ inside(my_office, marie)
ok, location of marie is the_office				
USUARIO: ***all drinks are in the fridge***	**Object:** **Action:** **Location:**	all drinks = apple_juice, root_beer are = is in the fridge = the_fridge	(ACTION: say_listen MSG: "ok, apple_juice and root_beer are in the_fridge"	+ inside(root_beer, the_fridge) + inside(apple_juice, the_fridge)
ok, apple_juice and root_beer are in the fridge				
USUARIO: ***juice is cold***	**Object:** **Action:** **Attribute:**	juice = juice are = is cold = cold	(ACTION: say_listen MSG: "ok, juice is cold"	+ temperature(juice, cold)
ok, juice is cold				
USUARIO: ***give a cold drink to the woman in the blue room***	**Action:** **Object:** **Origen:** **Destination:**	give = fetch a cold drink = apple_juice consult(location,apple_juice) to the woman in the blue room = marie	(ACTION: fetch Object: apple_juice From: front_kitchen To: marie)	+performing(robot, fetch) - location(robot, *)

Figura 7.5 Fragmento de conversación para ingresar y consultar información.

En el cuadro 7.5 se muestra el registro anotado de un fragmento de la conversación que el sistema entabla. En el cuadro se muestran los parámetros junto con las palabras o

la consulta a la base de conocimiento a partir de las cuales se aterrizaron. En la columna *Comando al planeador* se muestra la expresión generada para enviar al planeador de acciones. En la última columna se muestran los nodos que se agregan o se eliminan a la ontología, los cuales se integran inmediatamente a la base de conocimiento.

Como se observa en la conversación, los patrones de interpretación permiten definir la forma de interpretar enunciados al mismo tiempo que consultan y amplían el conocimiento en la ontología. Además, cada interpretación provoca un comando al planeador para ser ejecutado por el robot. Con esta configuración, el sistema procesa adecuadamente frases como:

- *grape_juice is an instance of juice*
- *look for drinks in the table that is in the kitchen and bring them*
- *who is the person in the livingroom?*
- *bring me something to eat*

Lo anterior en su conjunto intenta cubrir el lenguaje que usualmente se requiere en la competencia *Robocup@home*.

7.2 Pruebas con planeación automatizada en tareas tipo Robocup

El proceso de interpretación permite estipular un comando al planeador de acciones, el cual debe contener toda la información necesaria para iniciar el proceso de planeación automatizada. La ejecución de tareas primitivas va por cuenta de módulos externos con los cuales el planeador se debe comunicar cuando sea necesario. Para la primera sección de pruebas presentadas, el programa del planeador de acciones simula la exitosa ejecución de las tareas primitivas agregando y quitando nodos de la ontología, acorde a los efectos que cada tarea primitiva debe tener en el mundo.

Para los experimentos de la prueba *Endurance General Purpose Service Robot* (EGPSR), los efectos sobre el mundo se simularon con la representación simbólica de un departamento con objetos, personas y lugares, de forma similar a las competencias. Por otro lado, hice pruebas usando el robot *Justina* del laboratorio de Biorobótica. Sin embargo, únicamente se probaron órdenes sencillas que requirieron ejecutar una sola tarea primitiva.

7.2.1 Prueba simulada EGPSR tipo 1

Las oraciones del tipo 1 de la prueba EGPSR son tres órdenes con parámetros concisos y el modelo del mundo no contiene información contradictoria que evite al robot ejecutar la orden. Para las pruebas presentadas, cada una de las tres órdenes debe enunciarse por separado.

Orden en lenguaje natural	Comandos generados	Secuencia de acciones	Cambios en el mundo
grab the cold drink that is in kitchen table	**(action TAKEOBJECT object apple_juice location kitchen 0 0)**	Navegation module - move(robot, from: living room, to: kitchen)	- in(justina, livingroom) + in(justina, kitchen)
		Navegation module - aproach(to: fridge1, in: kitchen)	+ attention(justina,fridge_1)
		Recognition module - look_for(class: juice, in: fridge1)	+ graspable(apple_juice, yes)
		Arm module - grasp(item: apple_juice)	- in(apple_juice,fridge_1) - attention(justina,fridge_1) - graspable(apple_juice, yes) + carry(justina, apple_juice)

Figura 7.6 Ejemplo de un plan para acciones del tipo 1 en la prueba EGPSR en Robocup@home.

En la figura 7.6 se muestran algunas órdenes de esta categoría junto con su inter-

pretación y el plan producido por el sistema. El comando generado corresponde con la sintaxis y tipos de parámetros del lenguaje con el que el planeador describe las tareas pendientes en la exploración de redes jerárquicas de tareas. En este ejemplo, la tarea se descompone en cuatro acciones ordenadas de alto nivel cuyos efectos en el mundo son precondiciones para acciones subsecuentes. Como estipulan las reglas de la competencia, en esta prueba no existe información contradictoria en el estado del mundo por lo que el plan se genera tal cual describe la red de tareas *take_object*.

De manera similar, se diseñaron otras redes jerárquicas de tareas que en su conjunto permiten generar planes para órdenes como:

- Mover objetos de un lugar a otro.
- Entregar un mensaje verbal a una persona.
- Ir a un lugar o con un objeto.
- Tomar objetos.

7.2.2 Prueba simulada EGPSR tipo 2

Las oraciones de tipo 2 de la prueba EGPSR son órdenes con información vaga o incompleta. El comité de la competencia plantea que la decisión de cómo manejar la información vaga o incompleta se deja a consideración del equipo. Por diseño, cuando el planeador debe aplicar la misma acción a un conjunto de objetos genera un comando independiente por cada elemento, indicando el orden de ejecución. Por lo tanto, al ingresar la oración *bring all cold drinks* el interprete de lenguaje genera dos órdenes al planeador de acciones. En el cuadro 7.7 se muestra la descripción de los planes generados.

El nombre y lugar de dichas bebidas y el destino final de la entrega se resuelven a nivel de interpretación de lenguaje, según el patrón de interpretación. El planeador genera lo siguiente:

Orden en lenguaje natural	Comandos generados	Secuencia de acciones	Cambios en el mundo
bring all cold drinks	**(action FETCH object apple_juice location kitchen destination livingroom)**	Navegation module - move(robot, from: living room, to: kitchen)	- in(justina, livingroom) + in(justina, kitchen)
		Navegation module - aproach(to: fridge1, in: kitchen)	+ attention(justina,fridge_1)
		Recognition module - look_for(class: juice, in: fridge1)	+ graspable(apple_juice, yes)
		Arm module - grasp(item: apple_juice)	- in(apple_juice,fridge_1) - attention(justina,fridge_1) - graspable(apple_juice, yes) + carry(justina, apple_juice)
		Navigation module - move robot from kitchen to livingroom	- in(justina, kitchen) + in(justina,livingroom)
		Speech generation module - generate_speech("i will drop the cargo apple_juice")	
		Arm module - put down item apple_juice	- carry(justina, apple_juice)
	(action FETCH object pils_1 location kitchen destination livingroom)	Navegation module - move(robot, from: living room, to: kitchen)	- in(justina, livingroom) + in(justina, kitchen)
		Navegation module - aproach(to: fridge1, in: kitchen)	+ attention(justina,fridge_1)
		Recognition module - look_for(class: beer, in: fridge1)	+ graspable(pils_1, yes)
		Arm module - grasp(item: pils_1)	- in(pils_1,fridge_1) - attention(justina,fridge_1) - graspable(pils_1, yes) + carry(justina, pils_1)
		Navigation module - move robot from kitchen to livingroom	- in(justina, kitchen) + in(justina,livingroom)
		Speech generation module - generate_speech("i will drop the cargo pils_1")	
		Arm module - put down item pils_1	- carry(justina, pils_1)

Figura 7.7 Ejemplo de un plan para acciones del tipo 2 en la prueba EGPSR en Robocup@home.

7.2.3 Prueba simulada EGPSR tipo 3

Las oraciones de tipo 3 de la prueba EGPSR son órdenes como las del tipo 1 y 2 pero existe alguna contradicción entre el mundo y la orden especificada. El robot debe resolver qué hacer en esa situación contradictoria.

En el modelo del mundo en este experimento, el robot está en el cuarto *living room* y se le pide ir a *kitchen*. No obstante, como la puerta que une ambos cuartos está cerrada no se puede derivar el hecho que ambos cuartos estén conectados, lo cual es una precondición para la ejecución de la tarea *go*. Esto provoca que el robot comience la ejecución de la tarea *go* pero al no cumplir con las precondiciones necesarias no acaba la actividad hasta que una *tarea posiblemente aplicable* se ejecute y vuelva aplicable la tarea *go*.

Orden en lenguaje natural	Comandos generados	Secuencia de acciones	Cambios en el mundo
go to the kitchen	**(action GO location kitchen 0 0)**	Idle (tarea go no aplicable)	
	Acción emergente	Arm module - open door in: kitchen_door in: kitchen	+ functional_status(kitchen_door,open) + connected(livingroom,kitchen)
		Navegation module - move(robot from: livingroom, to: kitchen)	- in(justina, livingroom) + in(justina, kitchen)

Figura 7.8 Ejemplo de un plan para acciones del tipo 3 en la prueba EGPSR en Robocup@home.

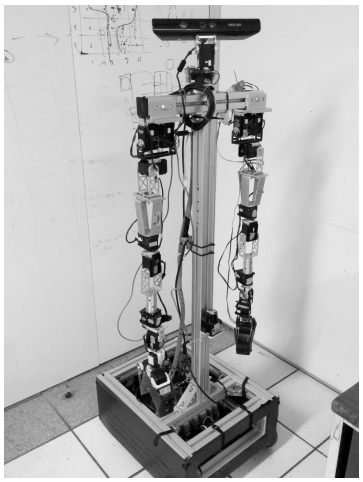
Como no se cumplen las precondiciones necesarias para mover al robot hacia la cocina, la acción falla en completarse, no se generan cambios en el mundo y el plan se interrumpe. El método *execute_emergent_open_door* se vuelve relevante, al poder proporcionar una precondición para un método en la agenda del planeador, y abre la puerta. Al añadir al modelo del mundo el hecho *functional_status(kitchen_door, open)* se dispara una regla de inferencia de conocimiento que agrega el hecho *connected(livingroom,kitchen)*. Así, el sistema llega a un contexto relevante para ejecutar la acción solicitada y se termina normalmente.

7.3 Pruebas con el robot Justina

Se hicieron dos tipos de pruebas sobre el robot *Justina*:

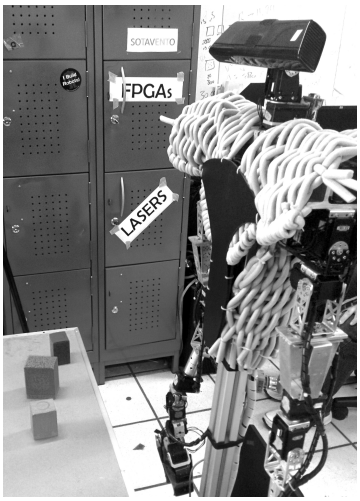
- Indicar direcciones al robot usando la interpretación de lenguaje natural y la planeación automatizada presentadas en esta tesis.
- Manipular bloques de colores usando la interpretación de lenguaje natural presentada en esta tesis y la planeación automatizada de una tercera parte.

Dentro de los subsistemas del robot *Justina*, el *action_planner* coordina los comportamientos de alto nivel y es el software que ejecuta las máquinas de estados jerárquicas



(a) Robot Justina.

(b) Demostración enunciado una orden al robot.



(c) Pruebas manipulando cubos mediante lenguaje natural.

que controlan el comportamiento global del robot. Para integrar este prototipo de software de interpretación de lenguaje natural y planeación automatizada, encapsulé su funcionalidad en una función que recibe una cadena de texto y manda mensajes directos a otros módulos durante el proceso de planeación. Dentro del módulo *action_planner* construimos una máquina de estados que tiene dos estados. En el primer estado el robot recibe una orden del usuario usando el reconocedor de voz de Microsoft. En el segundo estado se interpreta el enunciado con el módulo *lenguage_understanding_and_planning*. La máquina de estados no requiere estados adicionales porque el resto de los comportamientos del robot se ejecutan por comando directo a módulos externos conforme al plan generado automáticamente. Las pruebas con el robot usando el planeador automático propio únicamente incluyeron órdenes para mover al robot a puntos etiquetados en el mapa, como *go to the back of the workshop*. El planeador de acciones genera y comunica comandos para el módulo *motion_planner* para ejecutar tareas sobre el hardware.

Posteriormente, integré el interprete de lenguaje natural con un planeador automático desarrollado por *Adrián Revuelta*, otro tesista del Laboratorio de Biorobótica. Combinando ambos proyectos, fue posible controlar al robot para ejecutar órdenes como *put the red cube on top of the blue one*.

Capítulo 8

Conclusiones y trabajo futuro

Ahora es una época interesante para diseñar sistemas basados en conocimiento para lograr comunicar máquinas y humanos mediante lenguaje natural, debido a la demanda esperada de software que permita a personas sin preparación técnica controlar a un robot adecuadamente.

Diseñar interfaces humano-computadora usando lenguaje es un problema de distinta naturaleza respecto a otros problemas de procesamiento de lenguaje natural como la creación de resúmenes automáticos o traducción automática entre lenguajes naturales humanos. Mientras estos últimos funcionan mejor usando métodos estadísticos entrenados con miles o millones de ejemplos etiquetados, para diseñar una interfaz humano-computadora se necesitan tomar decisiones hechas a la medida de la aplicación y a las capacidades de la máquina que se controla. Por lo anterior, diseñar interfaces humano-computadora mediante un lenguaje natural controlado parece una solución viable y conveniente. La definición del lenguaje natural controlado aceptado por el sistema mediante los patrones de interpretación propuestos, combina inferencia sobre la información de una ontología, información sintáctica del idioma, resolución de frases nominales y preposicionales, restricciones sobre el tipo semántico de cada rol en el enunciado y acceso a funciones externas.

El planeador automático propuesto sigue la línea de la planeación automática clásica y propone la descomposicón de redes jerárquicas de tareas y la ejecución de tareas primitivas usando módulos externos como una forma expresiva y fácil de diseñar para administrar los comportamientos del robot doméstico. Al estar conectado a una base de conocimiento, el planeador tiene acceso a toda la información explícita e implícita del modelo del mundo, permitiendo expresar precondiciones y efectos sobre la ejecución de tareas a distintos niveles de abstracción.

Logré modelar la representación simbólica del mundo, la información lingüística de las oraciones esperadas en las interacciones propuestas y las capacidades del robot en

el planeador de acciones para que funcionaran adecuadamente en el micromundo de un robot de servicio doméstico. La integración del sistema con el robot *Justina* se llevó a cabo con éxito. La particiapación del robot en la competencia *RockIn@home*, celebrada en Toulouse, Francia, dejó como resultado un segundo lugar en la prueba *Speech Understanding*, en la cual utilizamos parte del software descrito en este trabajo. Además, logramos integrar el software de interpretación de lenguaje y planeación automatizada dentro de la arquitectura del robot *Justina* lo suficiente para mover al robot con órdenes por voz como *go to the back of the workshop*. Además, en conjunto con un planeador automático desarrollado por otro tesista del laboratorio, logramos que el robot manipulara cubos de colores, al vocalizar oraciones similares a las del corpus SEMEVAL 2014 tarea 6.

Al procesar el los enunciados del corpus SEMEVAL 2014 tarea 6 se tiene una forma de comparación directa con otros métodos de procesamiento de lenguaje natural, en su mayoría estadísticos. A pesar de que los resultados se comparan con métodos de distinta naturaleza y compromisos, se incluyó el análisis del corpus para experimentar con la capacidad de reconfiguración del sistema para producir diferentes expresiones de significado. Los resultados se consideran adecuados a lo esperado al compararse con el rendimiento de una gramática hecha a mano. Sin embargo, esta prueba no evalúa todas las características del sistema presentado, al no usar información del estado del mundo para interpretar las oraciones y no involucrar al planeador de acciones.

Incluso en interacciones simples como las que se presentan en este trabajo, la elección de distintos algoritmos y el proceso de evaluación tienen espacio para mejorar. Se puede trabajar en distintas actividades para aumentar la calidad de la solución, estas incluyen:

- Realizar corpus anotados sobre interacciones humano-robot y humano-humano en actividades domésticas cooperativas. A falta de un estándar sobre lo que las personas comunes hacen o esperan hacer con un robot doméstico, el diseño del alcance de las interacciones tiene un fuerte componente de especulación e introspección, provocando un sesgo sobre el planteamiento de los requisitos del sistema. Además, contar con un corpus específico en este contexto permitiría usar métodos estadísticos para algunas decisiones en el proceso de interpretación como la desambiguación léxica o la identificación sintáctica de frases nominales.

- Integrar lo antes posible el robot real en el proceso de evaluación del planeador de tareas. Se deben hacer pruebas reales en sitio para evitar encontrar limitaciones, comportamientos y casos no previstos.

- Integrar mecanismos de verificación sobre el conocimiento de la ontología. Se debe considerar la inclusión de un mecanismo robusto de monitoreo y mantenimiento sobre la información de la ontología para evitar llevar a la ontología a una configuración que produzca información incorrecta.

- Integrar otros tipos de inferencia en la base de conocimiento. Usando lógica formal y procesos probabilísticos como cadenas de markov se puede proveer al robot de mecanismos para aumentar la recopilación de información implícita que sostiene la ontología, como razonar sobre tiempo y eventos, sobre la interacción con el usuario a nivel de discurso o incluso inferir las posibles intenciones futuras del usuario.
- Integrar un manejador de diálogo robusto que permita, entre otras cosas, resolver fenómenos de anáfora y elipsis.

Capítulo 9

Apéndice

Muestra del corpus semeval 2014

El corpus se publicó por un grupo de inteligencia artificial de la universidad de Leeds en el Reino Unido proponiendo la tarea compartida *Supervised Semantic Parsing of Robotic Spatial Commands*, el corpus provee anotaciones semánticas de 3,409 comandos espaciales (41,158 palabras) para manipular figuras geométricas de distintos colores en un espacio tridimensional. Los enunciados provienen del uso de un juego en línea diseñado para capturar distintas escenas en un tablero tridimensional con ambigüedad espacial, confusión de color, marcas en la escena y grupos de figuras grandes y pequeños, los usuarios recibieron dos escenas distintas y escribieron comandos a un robot hipotético para lograr que ambas escenas sean iguales. El treebank mapea las oraciones a un lenguaje formal llamado Robot Control Language RCL [50].

Para más información consultar http://alt.qcri.org/semeval2014/task6/

Los primeros 100 enunciados del corpus son:

Enunciado de entrada	**Comando RCL**
19 8 place green pyramid on top of red brick	19 rcl (event: (action: move (token: 1)) (entity: (color: green (token: 2)) (type: prism (token: 3))) (destination: (spatial-relation: (relation: above (token: 4 6)) (entity: (color: red (token: 7)) (type: cube (token: 8))))))
28 187 place the red pyramid sitting on top of the red brick on top of the yellow one	28 rcl (event: (action: move (token: 1)) (entity: (color: red (token: 3)) (type: prism (token: 4)) (spatial-relation: (relation: above (token: 5 8)) (entity: (id: 1) (color: red (token: 10)) (type: cube (token: 11))))) (destination: (spatial-relation: (relation: above (token: 12 14)) (entity: (color: yellow (token: 16)) (type: type-reference (token: 17)) (reference-id: 1)))))
34 795 Move the blue block on top of the grey block.	34 rcl (event: (action: move (token: 1)) (entity: (color: blue (token: 3)) (type: cube (token: 4))) (destination: (spatial-relation: (relation: above (token: 5 7)) (entity: (color: gray (token: 9)) (type: cube (token: 10))))))

48 820 Move the yellow tetrahedron on top of the red blocks in the corner nearest to the yellow tetrahedron.	48 rcl (event: (action: move (token: 1)) (entity: (color: yellow (token: 3)) (type: prism (token: 4))) (destination: (spatial-relation: (relation: above (token: 5 7)) (entity: (color: red (token: 9)) (type: cube-group (token: 10)) (spatial-relation: (relation: within (token: 11)) (entity: (type: corner (token: 13)) (spatial-relation: (relation: nearest (token: 14)) (entity: (color: yellow (token: 17)) (type: prism (token: 18))))))))))
57 785 Take the white block on top of the dark blue blocks and move it onto the white block that is on top of the grey blocks.	57 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: white (token: 3)) (type: cube (token: 4)) (spatial-relation: (relation: above (token: 5 7)) (entity: (color: blue (token: 9 10)) (type: cube-group (token: 11)))))) (event: (action: drop (token: 13)) (entity: (type: reference (token: 14)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 15)) (entity: (color: white (token: 17)) (type: cube (token: 18)) (spatial-relation: (relation: above (token: 21 23)) (entity: (color: gray (token: 25)) (type: cube-group (token: 26))))))))
78 145 place the blue brick on top of the red brick	78 rcl (event: (action: move (token: 1)) (entity: (color: blue (token: 3)) (type: cube (token: 4))) (destination: (spatial-relation: (relation: above (token: 5 7)) (entity: (color: red (token: 9)) (type: cube (token: 10))))))
81 898 Move the yellow tetrahedron on top of the nearest green block.	81 rcl (event: (action: move (token: 1)) (entity: (color: yellow (token: 3)) (type: prism (token: 4))) (destination: (spatial-relation: (relation: above (token: 5 7)) (entity: (indicator: nearest (token: 9)) (color: green (token: 10)) (type: cube (token: 11))))))
84 45 Pick up the green tetrahedron.	84 rcl (event: (action: take (token: 1 2)) (entity: (color: green (token: 4)) (type: prism (token: 5))))
131 300 Take the light blue prism that is on top of the dark blue block and move it on top of the white block.	131 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: cyan (token: 3 4)) (type: prism (token: 5)) (spatial-relation: (relation: above (token: 8 10)) (entity: (color: blue (token: 12 13)) (type: cube (token: 14)))))) (event: (action: drop (token: 16)) (entity: (type: reference (token: 17)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 18 20)) (entity: (color: white (token: 22)) (type: cube (token: 23)))))))
185 147 Move the red prism on top of the blue cube.	185 rcl (event: (action: move (token: 1)) (entity: (color: red (token: 3)) (type: prism (token: 4))) (destination: (spatial-relation: (relation: above (token: 5 7)) (entity: (color: blue (token: 9)) (type: cube (token: 10))))))
196 897 place the yellow pyramid on top of the light grey brick	196 rcl (event: (action: move (token: 1)) (entity: (color: yellow (token: 3)) (type: prism (token: 4))) (destination: (spatial-relation: (relation: above (token: 5 7)) (entity: (color: white (token: 9 10)) (type: cube (token: 11))))))
233 644 Drop the grey block.	233 rcl (event: (action: drop (token: 1)) (entity: (color: gray (token: 3)) (type: cube (token: 4))))
264 974 Place blue block on top of single red block.	264 rcl (event: (action: drop (token: 1)) (entity: (color: blue (token: 2)) (type: cube (token: 3))) (destination: (spatial-relation: (relation: above (token: 4 6)) (entity: (indicator: individual (token: 7)) (color: red (token: 8)) (type: cube (token: 9))))))
295 729 Place green block on top of blue block.	295 rcl (event: (action: move (token: 1)) (entity: (color: green (token: 2)) (type: cube (token: 3))) (destination: (spatial-relation: (relation: above (token: 4 6)) (entity: (color: blue (token: 7)) (type: cube (token: 8))))))
306 729 place green brick on top of the blue one	306 rcl (event: (action: move (token: 1)) (entity: (id: 1) (color: green (token: 2)) (type: cube (token: 3))) (destination: (spatial-relation: (relation: above (token: 4 6)) (entity: (color: blue (token: 8)) (type: type-reference (token: 9)) (reference-id: 1)))))
336 998 Place current block on top of blue block.	336 rcl (event: (action: drop (token: 1)) (entity: (type: cube (token: 3))) (destination: (spatial-relation: (relation: above (token: 4 6)) (entity: (color: blue (token: 7)) (type: cube (token: 8))))))

358 708 pick the red block and put it above the yellow block	358 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: red (token: 3)) (type: cube (token: 4)))) (event: (action: drop (token: 6)) (entity: (type: reference (token: 7)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 8)) (entity: (color: yellow (token: 10)) (type: cube (token: 11)))))))
400 54 pick up the white block that is on the top of green block in the corner and place it on the white single block	400 rcl (sequence: (event: (action: take (token: 1 2)) (entity: (id: 1) (color: white (token: 4)) (type: cube (token: 5)) (spatial-relation: (relation: above (token: 8 11)) (entity: (color: green (token: 12)) (type: cube (token: 13)) (spatial-relation: (relation: within (token: 14)) (entity: (type: corner (token: 16)))))))) (event: (action: drop (token: 18)) (entity: (type: reference (token: 19)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 20)) (entity: (indicator: individual (token: 23)) (color: white (token: 22)) (type: cube (token: 24)))))))
403 57 pick the red block and put it above the purple block	403 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: red (token: 3)) (type: cube (token: 4)))) (event: (action: drop (token: 6)) (entity: (type: reference (token: 7)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 8)) (entity: (color: magenta (token: 10)) (type: cube (token: 11)))))))
409 45 Pick up green pyramid.	409 rcl (event: (action: take (token: 1 2)) (entity: (color: green (token: 3)) (type: prism (token: 4))))
415 190 Place grey pyramid on top of grey block.	415 rcl (event: (action: move (token: 1)) (entity: (color: gray (token: 2)) (type: prism (token: 3))) (destination: (spatial-relation: (relation: above (token: 4 6)) (entity: (color: gray (token: 7)) (type: cube (token: 8))))))
424 340 put the blue block above the yellow block	424 rcl (event: (action: drop (token: 1)) (entity: (color: blue (token: 3)) (type: cube (token: 4))) (destination: (spatial-relation: (relation: above (token: 5)) (entity: (color: yellow (token: 7)) (type: cube (token: 8))))))
430 989 Pick up nearest block.	430 rcl (event: (action: take (token: 1 2)) (entity: (indicator: nearest (token: 3)) (type: cube (token: 4))))
503 998 put the red block above the blue block	503 rcl (event: (action: drop (token: 1)) (entity: (color: red (token: 3)) (type: cube (token: 4))) (destination: (spatial-relation: (relation: above (token: 5)) (entity: (color: blue (token: 7)) (type: cube (token: 8))))))
570 549 place the red brick on top of the blue brick	570 rcl (event: (action: move (token: 1)) (entity: (color: red (token: 3)) (type: cube (token: 4))) (destination: (spatial-relation: (relation: above (token: 5 7)) (entity: (color: blue (token: 9)) (type: cube (token: 10))))))
603 701 place blue brick on top of the green brick	603 rcl (event: (action: move (token: 1)) (entity: (color: blue (token: 2)) (type: cube (token: 3))) (destination: (spatial-relation: (relation: above (token: 4 6)) (entity: (color: green (token: 8)) (type: cube (token: 9))))))
639 596 place the green pyramid down	639 rcl (event: (action: drop (token: 1)) (entity: (color: green (token: 3)) (type: prism (token: 4))))
660 793 place the blue block on top of the green block	660 rcl (event: (action: move (token: 1)) (entity: (color: blue (token: 3)) (type: cube (token: 4))) (destination: (spatial-relation: (relation: above (token: 5 7)) (entity: (color: green (token: 9)) (type: cube (token: 10))))))
728 186 place the turquoise pyramid on top of the turquoise block	728 rcl (event: (action: move (token: 1)) (entity: (color: cyan (token: 3)) (type: prism (token: 4))) (destination: (spatial-relation: (relation: above (token: 5 7)) (entity: (color: cyan (token: 9)) (type: cube (token: 10))))))
827 9 Pick the yellow block on top of the red block and place it on top of the green block.	827 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: yellow (token: 3)) (type: cube (token: 4)) (spatial-relation: (relation: above (token: 5 7)) (entity: (color: red (token: 9)) (type: cube (token: 10)))))) (event: (action: drop (token: 12)) (entity: (type: reference (token: 13)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 14 16)) (entity: (color: green (token: 18)) (type: cube (token: 19)))))))

833 751 Take the Green Pyramid and put it on the yellow box	833 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: green (token: 3)) (type: prism (token: 4)))) (event: (action: drop (token: 6)) (entity: (type: reference (token: 7)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 8)) (entity: (color: yellow (token: 10)) (type: cube (token: 11)))))))
864 365 place the green pyramid on top of the blue block in the far left corner	864 rcl (event: (action: move (token: 1)) (entity: (color: green (token: 3)) (type: prism (token: 4))) (destination: (spatial-relation: (relation: above (token: 5 7)) (entity: (color: blue (token: 9)) (type: cube (token: 10)) (spatial-relation: (relation: within (token: 11)) (entity: (indicator: front (token: 13)) (indicator: left (token: 14)) (type: corner (token: 15))))))))
924 963 Grab the red pyramid.	924 rcl (event: (action: take (token: 1)) (entity: (color: red (token: 3)) (type: prism (token: 4))))
951 424 Put the yellow pyramid on top of the grey tower.	951 rcl (event: (action: move (token: 1)) (entity: (color: yellow (token: 3)) (type: prism (token: 4))) (destination: (spatial-relation: (relation: above (token: 5 7)) (entity: (color: gray (token: 9)) (type: stack (token: 10))))))
956 503 Grab the yellow pyramid.	956 rcl (event: (action: take (token: 1)) (entity: (color: yellow (token: 3)) (type: prism (token: 4))))
1003 326 Move the green cube one square to the left.	1003 rcl (event: (action: move (token: 1)) (entity: (color: green (token: 3)) (type: cube (token: 4))) (destination: (spatial-relation: (measure: (entity: (cardinal: 1 (token: 5)) (type: tile (token: 6)))) (relation: left (token: 9)))))
1023 591 Move the turquoise pyramid on top of blue block to the top of the other blue block.	1023 rcl (event: (action: move (token: 1)) (entity: (color: cyan (token: 3)) (type: prism (token: 4)) (spatial-relation: (relation: above (token: 5 7)) (entity: (color: blue (token: 8)) (type: cube (token: 9))))) (destination: (spatial-relation: (relation: above (token: 12 13)) (entity: (color: blue (token: 16)) (type: cube (token: 17))))))
1045 205 Move the green pyramid on top of the blue block to the top of the other blue block.	1045 rcl (event: (action: move (token: 1)) (entity: (color: green (token: 3)) (type: prism (token: 4)) (spatial-relation: (relation: above (token: 5 7)) (entity: (color: blue (token: 9)) (type: cube (token: 10))))) (destination: (spatial-relation: (relation: above (token: 13 14)) (entity: (color: blue (token: 17)) (type: cube (token: 18))))))
1088 918 Move the white cube on green , red cubes to the top of the blue cubes.	1088 rcl (event: (action: move (token: 1)) (entity: (color: white (token: 3)) (type: cube (token: 4)) (spatial-relation: (relation: above (token: 5)) (entity: (color: green (token: 6)) (color: red (token: 8)) (type: cube-group (token: 9))))) (destination: (spatial-relation: (relation: above (token: 12 13)) (entity: (color: blue (token: 15)) (type: cube-group (token: 16))))))
1090 153 Move the yellow tetrahedron from the top of the turquoise building block to the top of the grey building block.	1090 rcl (event: (action: move (token: 1)) (entity: (color: yellow (token: 3)) (type: prism (token: 4)) (spatial-relation: (relation: above (token: 7 8)) (entity: (color: cyan (token: 10)) (type: cube (token: 12))))) (destination: (spatial-relation: (relation: above (token: 15 16)) (entity: (color: gray (token: 18)) (type: cube (token: 20))))))
1094 259 Move pink pyramid to the top of the leftmost red block.	1094 rcl (event: (action: move (token: 1)) (entity: (color: magenta (token: 2)) (type: prism (token: 3))) (destination: (spatial-relation: (relation: above (token: 6 7)) (entity: (indicator: leftmost (token: 9)) (color: red (token: 10)) (type: cube (token: 11))))))
1106 549 Move red cube on top of blue cube.	1106 rcl (event: (action: move (token: 1)) (entity: (color: red (token: 2)) (type: cube (token: 3))) (destination: (spatial-relation: (relation: above (token: 4 6)) (entity: (color: blue (token: 7)) (type: cube (token: 8))))))
1111 503 Remove the yellow pyramid from the board.	1111 rcl (event: (action: take (token: 1)) (entity: (color: yellow (token: 3)) (type: prism (token: 4))))

1113 410 Pick up the gray triangle and place it on top of the red cube which is placed on a yellow cube.	1113 rcl (sequence: (event: (action: take (token: 1 2)) (entity: (id: 1) (color: gray (token: 4)) (type: prism (token: 5)))) (event: (action: drop (token: 7)) (entity: (type: reference (token: 8)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 9 11)) (entity: (color: red (token: 13)) (type: cube (token: 14)) (spatial-relation: (relation: above (token: 17 18)) (entity: (color: yellow (token: 20)) (type: cube (token: 21))))))))))
1119 916 Pick up the white cube on green and red blocks and place it on top of the blue blocks	1119 rcl (sequence: (event: (action: take (token: 1 2)) (entity: (id: 1) (color: white (token: 4)) (type: cube (token: 5)) (spatial-relation: (relation: above (token: 6)) (entity: (color: green (token: 7)) (color: red (token: 9)) (type: cube-group (token: 10)))))) (event: (action: drop (token: 12)) (entity: (type: reference (token: 13)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 14 16)) (entity: (color: blue (token: 18)) (type: cube-group (token: 19)))))))
1153 789 Pick the which block on top of the sky blue block and place it over the red block which is near to bunch of blue blocks.	1153 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (type: cube (token: 4)) (spatial-relation: (relation: above (token: 5 7)) (entity: (color: cyan (token: 9 10)) (type: cube (token: 11)))))) (event: (action: drop (token: 13)) (entity: (type: reference (token: 14)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 15)) (entity: (color: red (token: 17)) (type: cube (token: 18)) (spatial-relation: (relation: nearest (token: 21)) (entity: (color: blue (token: 25)) (type: cube-group (token: 26))))))))
1161 157 Pick up the yellow triangle placed on top of the green cube and place it on top of the red cube	1161 rcl (sequence: (event: (action: take (token: 1 2)) (entity: (id: 1) (color: yellow (token: 4)) (type: prism (token: 5)) (spatial-relation: (relation: above (token: 6 9)) (entity: (color: green (token: 11)) (type: cube (token: 12)))))) (event: (action: drop (token: 14)) (entity: (type: reference (token: 15)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 16 18)) (entity: (color: red (token: 20)) (type: cube (token: 21)))))))
1169 852 Pick up red block which is on top of the grey block and put this on top of the blue block	1169 rcl (sequence: (event: (action: take (token: 1 2)) (entity: (id: 1) (color: red (token: 3)) (type: cube (token: 4)) (spatial-relation: (relation: above (token: 7 9)) (entity: (color: gray (token: 11)) (type: cube (token: 12)))))) (event: (action: drop (token: 14)) (entity: (type: reference (token: 15)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 16 18)) (entity: (color: blue (token: 20)) (type: cube (token: 21)))))))
1173 965 Pick up the blue prism	1173 rcl (event: (action: take (token: 1 2)) (entity: (color: blue (token: 4)) (type: prism (token: 5))))
1206 178 Place yellow prism on top of the red cube	1206 rcl (event: (action: drop (token: 1)) (entity: (color: yellow (token: 2)) (type: prism (token: 3))) (destination: (spatial-relation: (relation: above (token: 4 6)) (entity: (color: red (token: 8)) (type: cube (token: 9))))))
1211 438 Move pink pyramid on top of grey block to the top of yellow block.	1211 rcl (event: (action: move (token: 1)) (entity: (color: magenta (token: 2)) (type: prism (token: 3)) (spatial-relation: (relation: above (token: 4 6)) (entity: (color: gray (token: 7)) (type: cube (token: 8))))) (destination: (spatial-relation: (relation: above (token: 11 12)) (entity: (color: yellow (token: 13)) (type: cube (token: 14))))))
1220 66 Pick the yellow block which is on top of a yellow block and move it above the leftmost blue block.	1220 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: yellow (token: 3)) (type: cube (token: 4)) (spatial-relation: (relation: above (token: 7 9)) (entity: (color: yellow (token: 11)) (type: cube (token: 12)))))) (event: (action: drop (token: 14)) (entity: (type: reference (token: 15)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 16)) (entity: (indicator: leftmost (token: 18)) (color: blue (token: 19)) (type: cube (token: 20)))))))

1229 939 Move the red block on top of the nearest blue block.	1229 rcl (event: (action: move (token: 1)) (entity: (color: red (token: 3)) (type: cube (token: 4))) (destination: (spatial-relation: (relation: above (token: 5 7)) (entity: (indicator: nearest (token: 9)) (color: blue (token: 10)) (type: cube (token: 11))))))
1232 976 Drop the red block on top of the left-most red block.	1232 rcl (event: (action: drop (token: 1)) (entity: (color: red (token: 3)) (type: cube (token: 4))) (destination: (spatial-relation: (relation: above (token: 5 7)) (entity: (indicator: leftmost (token: 9)) (color: red (token: 10)) (type: cube (token: 11))))))
1242 959 Move the pyramid on top of the cube.	1242 rcl (event: (action: move (token: 1)) (entity: (type: prism (token: 3))) (destination: (spatial-relation: (relation: above (token: 4 6)) (entity: (type: cube (token: 8))))))
1252 514 Pick the red pyramid which is on top of yellow brick and place it above the yellow block which is on its left.	1252 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: red (token: 3)) (type: prism (token: 4)) (spatial-relation: (relation: above (token: 7 9)) (entity: (color: yellow (token: 10)) (type: cube (token: 11)))))) (event: (action: drop (token: 13)) (entity: (type: reference (token: 14)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 15)) (entity: (color: yellow (token: 17)) (type: cube (token: 18)) (spatial-relation: (relation: above (token: 21)) (entity: (indicator: left (token: 23)) (type: region))))))))
1253 155 Move yellow pyramid on top of pink cubes to the top of red cube.	1253 rcl (event: (action: move (token: 1)) (entity: (color: yellow (token: 2)) (type: prism (token: 3)) (spatial-relation: (relation: above (token: 4 6)) (entity: (color: magenta (token: 7)) (type: cube-group (token: 8))))) (destination: (spatial-relation: (relation: above (token: 11 12)) (entity: (color: red (token: 13)) (type: cube (token: 14))))))
1259 707 Move the red cube on top of the other yellow cube.	1259 rcl (event: (action: move (token: 1)) (entity: (color: red (token: 3)) (type: cube (token: 4))) (destination: (spatial-relation: (relation: above (token: 5 7)) (entity: (color: yellow (token: 10)) (type: cube (token: 11))))))
1264 602 Pick the yellow block which is on top of red block and place it above the blue block.	1264 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: yellow (token: 3)) (type: cube (token: 4)) (spatial-relation: (relation: above (token: 7 9)) (entity: (color: red (token: 10)) (type: cube (token: 11)))))) (event: (action: drop (token: 13)) (entity: (type: reference (token: 14)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 15)) (entity: (color: blue (token: 17)) (type: cube (token: 18)))))))
1282 81 Pick the red block and place it on top of the highest leftmost green block.	1282 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: red (token: 3)) (type: cube (token: 4)))) (event: (action: drop (token: 6)) (entity: (type: reference (token: 7)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 8 10)) (entity: (indicator: leftmost (token: 13)) (color: green (token: 14)) (type: cube-group (token: 15)))))))
1287 340 Drop the blue cube on top of the yellow cube	1287 rcl (event: (action: drop (token: 1)) (entity: (color: blue (token: 3)) (type: cube (token: 4))) (destination: (spatial-relation: (relation: above (token: 5 7)) (entity: (color: yellow (token: 9)) (type: cube (token: 10))))))
1290 78 Put the grey block down on the left of the red block	1290 rcl (event: (action: drop (token: 1)) (entity: (color: gray (token: 3)) (type: cube (token: 4))) (destination: (spatial-relation: (relation: left (token: 8 9)) (entity: (color: red (token: 11)) (type: cube (token: 12))))))
1296 343 Pick up the blue cube placed on top of the red cube	1296 rcl (event: (action: take (token: 1 2)) (entity: (color: blue (token: 4)) (type: cube (token: 5)) (spatial-relation: (relation: above (token: 6 9)) (entity: (color: red (token: 11)) (type: cube (token: 12))))))

1330 206 pick the green pyramid above the blue block and put it on the other blue block	1330 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: green (token: 3)) (type: prism (token: 4)) (spatial-relation: (relation: above (token: 5)) (entity: (color: blue (token: 7)) (type: cube (token: 8)))))) (event: (action: drop (token: 10)) (entity: (type: reference (token: 11)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 12)) (entity: (color: blue (token: 15)) (type: cube (token: 16)))))))
1348 95 pick the red pyramid on the edge of the board and put it above the grey and blue parallelipiped	1348 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: red (token: 3)) (type: prism (token: 4)) (spatial-relation: (relation: above (token: 5)) (entity: (type: edge (token: 7)))))) (event: (action: drop (token: 12)) (entity: (type: reference (token: 13)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 14)) (entity: (color: gray (token: 16)) (color: blue (token: 18)) (type: stack (token: 19)))))))
1361 650 put the green block above the yellow parallelipiped	1361 rcl (event: (action: drop (token: 1)) (entity: (color: green (token: 3)) (type: cube (token: 4))) (destination: (spatial-relation: (relation: above (token: 5)) (entity: (color: yellow (token: 7)) (type: stack (token: 8))))))
1375 963 Lift the red prism.	1375 rcl (event: (action: take (token: 1)) (entity: (color: red (token: 3)) (type: prism (token: 4))))
1389 591 Take the light blue prism from the blue cube and put it on the other blue cube.	1389 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: cyan (token: 3 4)) (type: prism (token: 5)) (spatial-relation: (relation: above) (entity: (color: blue (token: 8)) (type: cube (token: 9)))))) (event: (action: drop (token: 11)) (entity: (type: reference (token: 12)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 13)) (entity: (color: blue (token: 16)) (type: cube (token: 17)))))))
1416 547 Take the red prism and put it on the green cube.	1416 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: red (token: 3)) (type: prism (token: 4)))) (event: (action: drop (token: 6)) (entity: (type: reference (token: 7)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 8)) (entity: (color: green (token: 10)) (type: cube (token: 11)))))))
1454 239 Pick up the green block.	1454 rcl (event: (action: take (token: 1 2)) (entity: (color: green (token: 4)) (type: cube (token: 5))))
1456 530 Take the green cube standing in the corner of the board and put it on the yellow one	1456 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: green (token: 3)) (type: cube (token: 4)) (spatial-relation: (relation: within (token: 5 6)) (entity: (type: corner (token: 8)))))) (event: (action: drop (token: 13)) (entity: (type: reference (token: 14)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 15)) (entity: (color: yellow (token: 17)) (type: type-reference (token: 18)) (reference-id: 1))))))
1478 395 pick the red block and put it on the blue block in the corner of the board	1478 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: red (token: 3)) (type: cube (token: 4)))) (event: (action: drop (token: 6)) (entity: (type: reference (token: 7)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 8)) (entity: (color: blue (token: 10)) (type: cube (token: 11)) (spatial-relation: (relation: within (token: 12)) (entity: (type: corner (token: 14))))))))))
1490 8 place the green pyramid on red brick	1490 rcl (event: (action: move (token: 1)) (entity: (color: green (token: 3)) (type: prism (token: 4))) (destination: (spatial-relation: (relation: above (token: 5)) (entity: (color: red (token: 6)) (type: cube (token: 7))))))
1491 699 Pick the green block and place it on top of the nearest blue block.	1491 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: green (token: 3)) (type: cube (token: 4)))) (event: (action: drop (token: 6)) (entity: (type: reference (token: 7)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 8 10)) (entity: (indicator: nearest (token: 12)) (color: blue (token: 13)) (type: cube (token: 14)))))))

1510 84 Take the red cube and place it in the far right corner of the board	1510 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: red (token: 3)) (type: cube (token: 4)))) (event: (action: drop (token: 6)) (entity: (type: reference (token: 7)) (reference-id: 1)) (destination: (spatial-relation: (relation: within (token: 8)) (entity: (indicator: front (token: 10)) (indicator: right (token: 11)) (type: corner (token: 12)))))))
1545 20 Drop the green pyramid	1545 rcl (event: (action: drop (token: 1)) (entity: (color: green (token: 3)) (type: prism (token: 4))))
1547 235 lift the blue cube which is near the far left corner	1547 rcl (event: (action: take (token: 1)) (entity: (color: blue (token: 3)) (type: cube (token: 4)) (spatial-relation: (relation: nearest (token: 7)) (entity: (indicator: front (token: 9)) (indicator: left (token: 10)) (type: corner (token: 11))))))
1573 921 pick the blue block above the yellow parallelipiped and put it above the red and green parallelipiped	1573 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: blue (token: 3)) (type: cube (token: 4)) (spatial-relation: (relation: above (token: 5)) (entity: (color: yellow (token: 7)) (type: stack (token: 8)))))) (event: (action: drop (token: 10)) (entity: (type: reference (token: 11)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 12)) (entity: (color: red (token: 14)) (color: green (token: 16)) (type: stack (token: 17)))))))
1579 510 place the yellow pyramid on red block	1579 rcl (event: (action: move (token: 1)) (entity: (color: yellow (token: 3)) (type: prism (token: 4))) (destination: (spatial-relation: (relation: above (token: 5)) (entity: (color: red (token: 6)) (type: cube (token: 7))))))
1588 885 Move the blue cube on top of the green cube	1588 rcl (event: (action: move (token: 1)) (entity: (color: blue (token: 3)) (type: cube (token: 4))) (destination: (spatial-relation: (relation: above (token: 5 7)) (entity: (color: green (token: 9)) (type: cube (token: 10))))))
1622 61 pick the purple pyramid and put it above the red block	1622 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: magenta (token: 3)) (type: prism (token: 4)))) (event: (action: drop (token: 6)) (entity: (type: reference (token: 7)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 8)) (entity: (color: red (token: 10)) (type: cube (token: 11)))))))
1625 396 pick the red block and put it above the grey block in the corner	1625 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: red (token: 3)) (type: cube (token: 4)))) (event: (action: drop (token: 6)) (entity: (type: reference (token: 7)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 8)) (entity: (color: gray (token: 10)) (type: cube (token: 11)) (spatial-relation: (relation: within (token: 12)) (entity: (type: corner (token: 14))))))))
1632 630 pick the yellow block above the red and turquoise parallelipiped and put it above the green block closest to the center of the board	1632 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: yellow (token: 3)) (type: cube (token: 4)) (spatial-relation: (relation: above (token: 5)) (entity: (color: red (token: 7)) (color: cyan (token: 9)) (type: stack (token: 10)))))) (event: (action: drop (token: 12)) (entity: (type: reference (token: 13)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 14)) (entity: (color: green (token: 16)) (type: cube (token: 17)) (spatial-relation: (relation: nearest (token: 18)) (entity: (indicator: center (token: 21)) (type: region))))))))
1635 501 Take the light blue pyramid.	1635 rcl (event: (action: take (token: 1)) (entity: (color: cyan (token: 3 4)) (type: prism (token: 5))))
1641 550 pick the red block above the blue block and put it above the white parallelipiped in the corner	1641 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: red (token: 3)) (type: cube (token: 4)) (spatial-relation: (relation: above (token: 5)) (entity: (color: blue (token: 7)) (type: cube (token: 8)))))) (event: (action: drop (token: 10)) (entity: (type: reference (token: 11)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 12)) (entity: (color: white (token: 14)) (type: stack (token: 15)) (spatial-relation: (relation: within (token: 16)) (entity: (type: corner (token: 18))))))))

1652 481 Pick the yellow pyramid in the center and put it on the blue block in the corner	1652 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: yellow (token: 3)) (type: prism (token: 4)) (spatial-relation: (relation: within (token: 5)) (entity: (indicator: center (token: 7)) (type: region))))) (event: (action: drop (token: 9)) (entity: (type: reference (token: 10)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 11)) (entity: (color: blue (token: 13)) (type: cube (token: 14)) (spatial-relation: (relation: within (token: 15)) (entity: (type: corner (token: 17))))))))
1663 888 pick the green block above the grey and red parallelipiped and put it on the red block	1663 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: green (token: 3)) (type: cube (token: 4)) (spatial-relation: (relation: above (token: 5)) (entity: (color: gray (token: 7)) (color: red (token: 9)) (type: stack (token: 10)))))) (event: (action: drop (token: 12)) (entity: (type: reference (token: 13)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 14)) (entity: (color: red (token: 16)) (type: cube (token: 17)))))))
1666 187 Take the red pyramid and place it on top of the yellow block.	1666 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: red (token: 3)) (type: prism (token: 4)))) (event: (action: drop (token: 6)) (entity: (type: reference (token: 7)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 8 10)) (entity: (color: yellow (token: 12)) (type: cube (token: 13)))))))
1670 228 pick the grey pyramid above the green block and put it above the red and yellow parallelipiped	1670 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: gray (token: 3)) (type: prism (token: 4)) (spatial-relation: (relation: above (token: 5)) (entity: (color: green (token: 7)) (type: cube-group (token: 8)))))) (event: (action: drop (token: 10)) (entity: (type: reference (token: 11)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 12)) (entity: (color: red (token: 14)) (color: yellow (token: 16)) (type: stack (token: 17)))))))
1703 749 Take the red prism and place it on top of the blue cube	1703 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: red (token: 3)) (type: prism (token: 4)))) (event: (action: drop (token: 6)) (entity: (type: reference (token: 7)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 8 10)) (entity: (color: blue (token: 12)) (type: cube (token: 13)))))))
1708 966 put the blue pyramid above the red block	1708 rcl (event: (action: drop (token: 1)) (entity: (color: blue (token: 3)) (type: prism (token: 4))) (destination: (spatial-relation: (relation: above (token: 5)) (entity: (color: red (token: 7)) (type: cube (token: 8))))))
1716 385 pick the turquoise pyramid above the grey and green parallelipiped	1716 rcl (event: (action: take (token: 1)) (entity: (color: cyan (token: 3)) (type: prism (token: 4)) (spatial-relation: (relation: above (token: 5)) (entity: (color: gray (token: 7)) (color: green (token: 9)) (type: stack (token: 10))))))
1747 850 Pick up the red cube placed on top of the gray cube and move it on top of the yellow cube	1747 rcl (sequence: (event: (action: take (token: 1 2)) (entity: (id: 1) (color: red (token: 4)) (type: cube (token: 5)) (spatial-relation: (relation: above (token: 6 9)) (entity: (color: gray (token: 11)) (type: cube (token: 12)))))) (event: (action: drop (token: 14)) (entity: (type: reference (token: 15)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 16 18)) (entity: (color: yellow (token: 20)) (type: cube (token: 21)))))))
1756 793 place the blue block above the green one	1756 rcl (event: (action: move (token: 1)) (entity: (id: 1) (color: blue (token: 3)) (type: cube (token: 4))) (destination: (spatial-relation: (relation: above (token: 5)) (entity: (color: green (token: 7)) (type: type-reference (token: 8)) (reference-id: 1)))))

1759 56 pick the red block above the blue block closest to the red and grey parallelipiped and put it above the blue block closest to the corner	1759 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: red (token: 3)) (type: cube (token: 4)) (spatial-relation: (relation: above (token: 5)) (entity: (color: blue (token: 7)) (type: cube (token: 8)) (spatial-relation: (relation: nearest (token: 9)) (entity: (color: red (token: 12)) (color: gray (token: 14)) (type: stack (token: 15)))))))) (event: (action: drop (token: 17)) (entity: (type: reference (token: 18)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 19)) (entity: (color: blue (token: 21)) (type: cube (token: 22)) (spatial-relation: (relation: nearest (token: 23)) (entity: (type: corner (token: 26)))))))))
1764 920 Pick the red block which is on top of the white block and move it on top of the green block.	1764 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: red (token: 3)) (type: cube (token: 4)) (spatial-relation: (relation: above (token: 7 9)) (entity: (color: white (token: 11)) (type: cube (token: 12)))))) (event: (action: drop (token: 14)) (entity: (type: reference (token: 15)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 16 18)) (entity: (color: green (token: 20)) (type: cube (token: 21)))))))
1768 57 pick the red block and put it above the purple block	1768 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: red (token: 3)) (type: cube (token: 4)))) (event: (action: drop (token: 6)) (entity: (type: reference (token: 7)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 8)) (entity: (color: magenta (token: 10)) (type: cube (token: 11)))))))
1774 497 pick the blue pyramid	1774 rcl (event: (action: take (token: 1)) (entity: (color: blue (token: 3)) (type: prism (token: 4))))
1780 939 Move the red block and place it on top of the blue block that is on top of a green block.	1780 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: red (token: 3)) (type: cube (token: 4)))) (event: (action: drop (token: 6)) (entity: (type: reference (token: 7)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 8 10)) (entity: (color: blue (token: 12)) (type: cube (token: 13)) (spatial-relation: (relation: above (token: 16 18)) (entity: (color: green (token: 20)) (type: cube (token: 21))))))))
1792 227 Move the grey pyramid from on top of the red block. Place it on top of the green block.	1792 rcl (sequence: (event: (action: take (token: 1)) (entity: (id: 1) (color: gray (token: 3)) (type: prism (token: 4)) (spatial-relation: (relation: above (token: 6 8)) (entity: (color: red (token: 10)) (type: cube (token: 11)))))) (event: (action: drop (token: 13)) (entity: (type: reference (token: 14)) (reference-id: 1)) (destination: (spatial-relation: (relation: above (token: 15 17)) (entity: (color: green (token: 19)) (type: cube (token: 20)))))))

Cuadro 9.1 Primeros 100 enunciados del corpus SEMEVAL 2014 tarea 6 con significado en RCL.

Bibliografía

[1] N. Fuchs, K. Kaljurand & G. Schneider. *Attempto Controlled English Meets the Challenges of Knowledge Representation, Reasoning, Interoperability and User Interfaces*. FLAIRS Conference (Vol. 12, pp. 664-669). 2006.

[2] C. Schlenoffa, Z. Kootballya & S. Foufoub. *Ontology-Based State Representation for Intention Recognition in Cooperative Human-Robot Environments*. Proceedings of the 2012 ACM Conference on Ubiquitous Computing (pp. 810-817). ACM. 2012.

[3] S. Ferrar & D. Langendoen. *A linguistic ontology for the semantic web*. Glot International. Vol 7, No 3. 2003.

[4] N. Fuchs, U. Schwertel & R. Schwitter. *Attempto Controlled English — Not Just Another Logic Specification Language*. Logic-Based Program Synthesis and Transformation. Springer Berlin Heidelberg. 1999.

[5] P. Clark, P. Harrison, T. Jenkinsl & T. Thompson. *Acquiring and Using World Knowledge Using a Restricted Subset of English*. FLAIRS Conference (pp. 506-511). 2005.

[6] K Trentelman. *Processable English:The Theory Behind the PENG System*. DSTO Formal Reports , Report number: DSTO-TR- 2301. 2009.

[7] R. Schwitter. *Controlled natural languages for knowledge representation*. Proceedings of the 23rd International Conference on Computational Linguistics: Posters (pp. 1113-1121). Association for Computational Linguistics. 2010.

[8] A. Popescu, O. Etzioni & H. Kautz. *Towards a theory of natural language interfaces to databases*. Proceedings of the 8th international conference on Intelligent user interfaces (pp. 149-157). ACM. 2003.

[9] Y. Wong, & R. Mooney *Learning for semantic parsing with statistical machine translation*. Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics (pp. 439-446). Association for Computational Linguistics. 2006.

[10] N. Nihalani, S. Silakari & M. Motwani *Natural language Interface for Database: A Brief review*. IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 2. 2011.

[11] D. Vogiatzis & V. Karkaletsis *A framework for human-robot interaction*. Proceedings of the 1st international conference on PErvasive Technologies Related to Assistive Environments (p. 10). ACM. 2008.

[12] R. Cantrell, M. Scheutz, P. Schermerhorn & X. Wu *Robust spoken instruction understanding for HRI*. Human-Robot Interaction (HRI), 2010 5th ACM/IEEE International Conference on (pp. 275-282). 2010.

[13] T. Kollar, S. Tellex, D. Roy & N. Roy *Toward understanding natural language directions*. Human-Robot Interaction (HRI), 2010 5th ACM/IEEE International Conference on (pp. 259-266). 2010.

[14] P. Yin *Natural Language Programming Based on Knowledge*. Artificial Intelligence and Computational Intelligence (AICI), 2010 International Conference on (Vol. 2, pp. 69-73). 2010.

[15] T. Breuer, G. Macedo, R. Hartanto, N. Hochgeschwender, D. Holz, F. Hegger & G. Kraetzschmar *Johnny: An autonomous service robot for domestic environments*. Journal of intelligent & robotic systems, 66(1-2), 245-272. 2012.

[16] S. Schiffer, A. Ferrein & G. Lakemeyer *Caesar: an intelligent domestic service robot*. Intelligent Service Robotics, 5(4), 259-273. 2012.

[17] S. Lemaignan, R. Ros, R. Alami & M. Beetz *Grounding the interaction: Anchoring situated discourse in everyday human-robot interaction*. International Journal of Social Robotics, 4(2), 181-199. 2012.

[18] L. Pineda et al. *SitLog: A Programming Language for Service Robot Tasks*. International Journal of Advanced Robotic Systems, vol. 10. 2013.

[19] M. Fleischman & D. Roy *Intentional context in situated natural language learning*. Proceedings of the Ninth Conference on Computational Natural Language Learning (pp. 104-111). Association for Computational Linguistics. 2005.

[20] R. Brachman & H. Levesque. *Knowledge representation and reasoning*. Elsevier. Estados unidos de América, 2004.

[21] H. Levesque & R. Reiter. *High level robotic control: beyond planning*. AIII Spring Symposium: Integrating Robotics Research: Taking the Next Big Leap, Vol. 37 1998.

[22] A. Newell. *High level robotic control: beyond planning*. Artificial intelligence magazine, Vol. 2, Num. 2. 1981.

[23] J. McCarthy. *From here to human-level AI*. Artificial intelligence magazine, Vol. 171, Num. 18. 2007.

[24] S. Lemaignan. *Grounding the Interaction: Knowledge Management for Interactive Robots*. Tesis doctoral, Université Paul Sabatier & Technische Universität München. 2012.

[25] H. Levesque & R. Brachman. *A Fundamental Tradeoff in Knowledge Representation and Reasoning*. Laboratory for Artificial Intelligence Research, Fairchild, Schlumberger. 1984.

[26] F. Harmelen & V. Lifschitz & B. Porter. *Handbook of Knowledge Representation*. Elsevier. Inglaterra. 2008.

[27] F. Baader & W. Nutt. *Basic description logics*. Description logic handbook. 2003.

[28] *CLIPS reference manual*. Manual en línea. 2007.

[29] B. Kuipers. *The Spatial Semantic Hierarchy*. Artificial intelligence magazine vol. 119, no 1. 2000.

[30] L. Pineda et al. *The Golem Team, RoboCup@home 2014*. 2014.

[31] M. Geert-Jan et al. *Situated Dialogue Processing for Human-Robot Interaction*. Cognitive Systems, COSMOS 8. Springer-Verlag Berlin Heidelberg. 2010.

[32] R. Schanck & G. Tesler *A conceptual parser for natural language*. Computer Science Department, School of Humanities and Sciences, Stanford University. 1969.

[33] R. Schanck. *The primitive acts of conceptual dependency*. Proceedings of the 1975 workshop on Theoretical issues in natural language processing. 1975.

[34] M Matamoros. *Análisis de extensibilidad, reestructuración y desempeño de software para robots móviles*. Tesis de maestría. Posgrado en Ciencia e Ingeniería de la Computación, UNAM. 2013.

[35] J Allen. *Natural language understanding*. The bejamin/Cummins publishing company. Estados Unidos de América. 1987.

[36] N. Indurkhya & F. Damerau. *Handbook of natural language processing*. CRC Press. Estados Unidos de América. 2010.

[37] D. Jurafsky & J. Martin. *Speech and language processing*. Prentice Hall. Estados Unidos de América. 1999.

[38] E. Ovchinnikova *Integration of world knowledge for natural language understanding*. Atlantis Press. Estados Unidos de América. 2012.

[39] M. Ghallab, D. Nau & P Traverso. *Automated planning. Theory and practice*. Elsevier. Estados Unidos de América. 2004.

[40] D. Off & J. Zhang *Hierarquical plan-based control in open-ended enviroments: cosidering knowledge acquisition opportunities*. Springer-Verlag. Communications in Computer and Information Science. Volumen 358. 2013.

[41] S Schneider & G Kraetzschmar. *RoCKIn@Home – A Competition for Domestic Service Robots -, rulebook*. 2014.

[42] J. Bos & T. Oka *A spoken language interface with a mobile robot*. Artificial Life and Robotics 11.1. 2007.

[43] V. Kulyukin. *Human-robot interaction through gesture-free spoken dialogue*. Autonomous Robots 16.3. 2004.

[44] R. Schwitter. *Representing knowledge in controlled natural language: a case study*. Knowledge-Based Intelligent Information and Engineering Systems. Springer Berlin Heidelberg. 2004.

[45] D. Misra. et al *Tell Me Dave: Context-Sensitive Grounding of Natural Language to Manipulation Instructions*. Robotics: Science and Systems (RSS). 2014.

[46] A. Ferrein, T. Niemueller & G. Lakemeyer. *Lessons Learnt from Developing the Embodied AI Platform CAESAR for Domestic Service Robotics*. Designing Intelligent Robots: Reintegrating AI II: AAAI Spring Symposium. 2013.

[47] S. Tellex, T. Kollar, S. Dickerson, M. Walter, A. Banerjee, S. Teller & R. Nicholas. *Approaching the Symbol Grounding Problem with Probabilistic Graphical Models*. AI Magazine vol. 32, no. 4. 2011.

[48] S. Tellex, T. Kollar, S. Dickerson, M. Walter, A. Banerjee, S. Teller & R. Nicholas. *Understanding Natural Language Commands for Robotic Navigation and Mobile Manipulation*. Proceedings of the National Conference on Artificial Intelligence. 2011.

[49] C. Matuszek, A. Pronobis, L. Zettlemoyer, & D. Fox. *Combining World and Interaction Models for Human-Robot Collaborations*. Proceedings of Workshops at the 27th AAAI Conference on Artificial Intelligence. 2013.

[50] K. Dukes. *Semantic Annotation of Robotic Spatial Commands*. In Language and Technology Conference (LTC). Poznan, Polonia. 2013.