Universidad Nacional de Córdoba
Facultad de Matemática, Astronomía y Física
Licenciatura en Ciencias de la Computación

Un sistema interactivo para la interpretación de especificaciones

Autor: Darío Garigliotti
Directora: Laura Alonso Alemany

Córdoba, 10 de marzo de 2014

Resumen

En este trabajo estudiamos el problema del tratamiento de una especificación de software expresada en lenguaje natural. Observamos y clasificamos fenómenos lingüísticos sobre un cuerpo de ejemplos de especificaciones. A su vez, exploramos algunos sistemas presentados en la literatura relacionada, identificando sus mejores características. A partir de las mismas, diseñamos e implementamos un sistema que interprete una especificación, expresada en un formato muy simple e informativo, y obtenga un sistema de transiciones etiquetadas. La estrategia de resolución combina un enfoque interactivo con heurísticas ad-hoc de decisión. Se enriquece la representación con el tratamiento de fenómenos semánticos. Varios lineamientos son ofrecidos para su extensión, en particular, hacia un modelo de anotación de ejemplos en el contexto educativo.

Clasificación:

D.2.1 Requirements/Specifications
I.2.1 Applications and Expert Systems
I.2.7 Natural Language Processing

Palabras Clave:

Especificación de requerimientos de software, requisitos funcionales, casos de uso, human-in-the-loop, crowdsourcing, interacción, recuperación de información, claves lingüísticas, expresiones ambigüas, detección temprana de la ambigüedad, sistema de transiciones etiquetadas, reglas de decisión, heurísticas ad-hoc, regeneración de lenguaje, anotación de ejemplos

Entre estos árboles que he inventado
y que no son árboles
estoy yo.

Roberto Bolaño. *La Universidad Desconocida.* La violencia es como la poesía.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Motivación y Objetivos

(Volver al Índice general)

Todo sistema de software es creado para dar respuesta a necesidades del mundo real. Por lo tanto, es fundamental que las formas en las que se represente esta realidad capturen con la mayor fidelidad posible las características del sistema deseado. El problema de la especificación de requerimientos es considerado como prioritario y decisivo durante el proceso de desarrollo de tales sistemas y la Ingeniería del Software ha propuesto diferentes técnicas y enfoques para resolverlo.

Más precisamente, el proceso de elicitación y análisis de requerimientos implica transformar conceptos del imaginario de un humano a representaciones sobre las que se pueda razonar, por ejemplo, especificaciones en lenguaje natural. Este proceso involucra generalmente un recurso humano capacitado, conocido como analista, que sostiene una serie de encuentros con el cliente para definir y progresivamente revisar, modificar y refinar los requisitos que se tienen para un sistema deseado.

Hay muchos factores involucrados en la construcción de textos de calidad. A la suficiencia de la cantidad y profundidad de las entrevistas de análisis, la experiencia del experto y su particular riqueza técnica en la confección de especificaciones en lenguaje natural, se le agrega uno trascendental. Una característica del lenguaje natural, que se manifiesta en particular en este tipo de especificaciones de interés, es que abundan las ambigüedades en todos los niveles de interpretación, desde la interpretación semántica a la de estructura sintáctica.

La ambigüedad es ubicua en el lenguaje natural. En particular resulta un problema considerable para requerimientos expresados en este nivel de lenguaje. Tal como se sintetiza en (Boyd, Zowghi, and Farroukh 2005), “la ambigüedad es propia de requerimientos de baja calidad”, luego es evidente que los errores en la interpretación de

las características del software deseado conducirá a errores más complejos y costosos de resolver en etapas posteriores, como las de diseño y de codificación. Como un claro indicador de esto, un error no resuelto en la etapa de requerimientos multiplica casi por quince su costo de resolución si se propaga hasta las etapas iniciales de testing (Mogyorodi 2001). Luego, la resolución de ambigüedades en la etapa de elicitación de requerimientos es muy conveniente incluso desde el punto de vista del costo del proyecto.

El objetivo de este trabajo es el de estudiar los aspectos relevantes del tratamiento de especificaciones de software en lenguaje natural, poniendo atención en el problema de las ambigüedades y su presencia en posibles puntos del problema. Adicionalmente, se desea diseñar e implementar un sistema concreto provisto de las características identificadas durante el análisis de otros sistemas y trabajos. El sistema analizaría una especificación de requerimientos funcionales de software expresada mediante un caso de uso y la interpretaría, obteniendo desde su información una representación de utilidad general.

Existen en la literatura relacionada otros trabajos y sistemas que abordan el problema de representar la información contenida en una especificación de requerimientos expresada en lenguaje natural – entre los revisados para este trabajo: (Fröhlich and Link 2000); (Pandita, Xiao, Zhong, Xie, Oney, and Paradkar 2012); (Lei, Long, Barzilay, and Rinard 2013). En general, proponen tratar la ambigüedad estableciendo restricciones en la forma de expresión de las especificaciones de requerimientos, por ejemplo, mediante la exigencia –al escritor del caso– de la utilización de un vocabulario controlado o de rígidas plantillas de casos. En el caso particular de las plantillas, varios conceptos involucrados, por ejemplo los de precondición y postcondición, escapan al usuario no educado y por lo tanto reducen el espacio de posibles interesados en especificar casos. Debe notarse también que las plantillas de casos de uso no aseguran automáticamente claridad en la expresión de los requisitos, sino que depende de la habilidad del escritor. Asimismo, no existen estándares obligatorios para escribir un caso de uso, sino convenciones ya estables que han sido exitosas.

A diferencia de éstos, nuestro sistema no exige sino un formato muy ágil de caso de uso e intenta resolver las mínimas ambigüedades necesarias para capturar con una buena precisión el comportamiento esperado. El resto de las ambigüedades permanecen como parte constitutiva de la expresión a través de todo el procesamiento. Como asociamos texto al grafo generado, es posible recuperar elementos del texto original para expresar una estructura de representación de conocimiento que se obtenga mediante alguna transformación aplicada al grafo original. De esta manera, las ambigüedades y vocabulario con los que el escritor dio forma a los requerimientos funcionales pueden

estar presentes en la expresión devuelta y ser así reconocidos fácilmente en lenguaje natural, aún se traten de diferentes usuarios, con posiblemente diferentes roles.

La característica fundamental de motivación y diseño del sistema que proponemos es el tratamiento que se hace de la naturaleza textual, y por tanto ambigua, de los casos de uso. El sistema presenta un enfoque de arquitectura *Human-in-the-loop*, es decir de interacción decisiva con el humano, y es de esta forma que la especificación puede ingresarse mediante una expresión muy flexible. Es el sistema el que sugiere interpretaciones y solicita alguna información adicional, mediante las cuales se intenta resolver las ambigüedades del lenguaje.

El desafío en el estudio y la construcción de este sistema que proponemos será, por tanto, el de desarrollar estrategias para obtener el máximo provecho del limitado cuerpo inicial de casos de uso de ejemplo y de la información que el humano provea, de forma que se preserve la flexibilidad deseada en el formato de expresión de entrada y una dinámica de interacción que optimice una mínima solicitud con máxima informatividad.

1.2. Estructura de la Tesis

(Volver al Índice general)

La estructura a continuación refleja la metodología que se sigue durante la realización del corriente trabajo. Esta tesis se organiza de la siguiente manera.

En el capítulo 2 se hace un relevamiento de otros trabajos y sistemas, de objetivos en mayor o menor medida similares al de este trabajo, que se han desarrollado en la investigación y la industria. Y se indican aspectos clave que destacan las características que pretendemos adoptar.

El enfoque *Human-in-the-Loop* propuesto como fundamental para los objetivos de este trabajo se desarrolla en el capítulo 3. En particular, se repasa la importancia del enfoque en varios contextos ya estudiados, y se redefinen objetivos a partir de su relación con esta estrategia de interacción.

El capítulo 4 presenta la arquitectura general del sistema. Las componentes involucradas, sus objetivos y tareas, son descritas en profundidad, así como el diseño de alto nivel de sus interfaces y estructuras de datos de comunicación. Desarrollamos también algunos conceptos fundamentales, y explicamos los aspectos decisivos de la estrategia de interacción que se propone.

A lo largo del capítulo 5 se puntualizan decisiones de mayor nivel de detalle en el diseño del sistema. Más aún, presentamos los fenómenos semánticos y lingüísticos que se propusieron estudiar en este trabajo, sus motivaciones y desafíos.

Finalmente, las conclusiones obtenidas en este trabajo se resumen en el capítulo

6. Asimismo, se presentan varias líneas de trabajo futuro que se avizoran promisorias para su estudio, implementación y/o evaluación a partir del trabajo aquí presentado.

Capítulo 2

Antecedentes Relevantes

(Volver al Índice general)

Presentamos un análisis de trabajos relevantes que seleccionamos de la literatura relacionada. Los mismos involucran diferentes tratamientos sobre especificaciones en lenguaje natural. A lo largo del análisis, pretendemos destacar aspectos positivos y negativos que se consideren de los sistemas desarrollados, al mismo tiempo, relevando un conjunto de posibles características para nuestro sistema.

2.1. De Texto a Sistemas de Transiciones. Restricciones de Entrada

Se aborda en (Gutiérrez 2005) el estudio de los sistemas de generación de pruebas o tests a partir de especificaciones funcionales. Realiza una exhaustiva comparación entre una docena de sistemas y trabajos publicados entre 1997 y 2004, observando parámetros como tipo de dato de entrada y de salida, uso de estándares, provisión de notación gráfica, criterio de cobertura, construcción del modelo, entre otros. Todos parten desde una especificación expresada como casos de uso en lenguaje natural, durante la etapa de elicitación de requisitos, para sistemas de utilidad general, pero se obtienen muchas representaciones diferentes como resultado: secuencias de transiciones, pruebas en lenguaje no formal, modelo de prueba, scripts de prueba ejecutables y combinaciones de los anteriores.

Ese trabajo concluye con resultados de impacto: las propuestas muestran un escaso conocimiento del problema, la documentación es de baja calidad, la estandarización es casi nula, y no hay sino escasas referencias a aplicaciones prácticas. En particular, uno de los sistemas allí estudiados, el ATCGDM de (Fröhlich and Link 2000) realiza una primera etapa de traducción de caso de uso en lenguaje natural a sistema de transi-

ciones; es en este sentido muy similar a nuestra propuesta. Pero el formato de entrada de datos que utiliza, si bien no establece un estándar invariable, sugiere fuertemente uno que es muy poco intuitivo para un humano no educado en los conceptos involucrados de especificación. Y aún para un analista con más experiencia resulta poco natural ya que el formato sugerido es el de una plantilla donde la secuencia dispone cada acción aislada en un bloque de operador - precondición - efecto. La salida al humano, orientada desde la concepción del sistema a resultar en casos de test, no es muy informativa y como paso intermedio muestra la secuencia de transiciones como una concatenación de acciones que llevan el estado inicial al objetivo. La naturaleza de una secuencia como esta queda más clara al saberse que utiliza luego fuertemente un lenguaje de planning para alcanzar los tests. La situación de una entrada expresada mediante un rígido formato se extiende a otros sistemas.

2.2. Un Universo de Sistemas

Una gran cantidad de sistemas han intentado dar tratamiento a una especificación en lenguaje natural; de nuevo, las representaciones de su resultado son variadas. Por ejemplo, (Price, Riloff, Zachary, Harvey, and Harvey 2000) presenta NaturalJava, un sistema que lleva especificaciones en lenguaje natural a código Java, con un interés en definir interfaces de usuario para que el código obtenido pueda ser examinado visualmente, simplificando la tarea de aprender sintaxis nueva y a la vez codificar correctamente.

En (Bajwa, Lee, and Bordbar 2012) se intenta obtener una restricción o fórmula en el lenguaje OCL –Object Constraint Language–, utilizado para establecer las restricciones de relaciones entre los elementos de un diagrama UML; en general esta restricción es obtenida manualmente. Este trabajo resulta interesante porque, si bien intenta una traducción automática desde lenguaje natural a una representación lógica, mucho más estricta y precisa que la nuestra, propone una arquitectura por etapas progresivamente de mayor abstracción, desde el nivel léxico al semántico. De todas formas, su representación es muy específica y dependiente del lenguaje OCL.

No sólo se han intentado resolver especificaciones de requisitos funcionales de software, sino también otras especificaciones en lenguaje natural. Tal es el caso de sistemas como el presentado en (Lei, Long, Barzilay, and Rinard 2013), que parte de la especificación del formato de los archivos de entrada de un programa dado para generar parsers de entrada, o el que se describe en (Pandita, Xiao, Zhong, Xie, Oney, and Paradkar 2012), donde la entrada es una interfaz de programación de aplicaciones descrita en un documento o manual de uso de un lenguaje y se intenta verificar la corrección del uso

de esas interfaces por parte de módulos concretos.

2.3. Enfoque Sistemático de las Ambigüedades

Más allá de las debilidades destacadas para la mayoría de estos sistemas, existe un problema común que muchos trabajos mencionados han siquiera mencionado: la ambigüedad inherente del lenguaje natural. Entendida como la posibilidad de razonablemente interpretar algo en más de una forma, la ambigüedad es un fenómeno que muchas veces escapa de la consideración de los humanos por estar tan habituados a su presencia y a utilizar mecanismos de la inteligencia durante la comunicación para desambiguar.

Como bien se expone en (Kiyavitskaya, Zeni, Mich, and Berry 2007), el humano practica lo que se conoce como desambiguación inconsciente, ignorando muchas interpretaciones sintácticamente posibles de una expresión, mediante el uso de criterios intuitivos que pueden generalizar muchos fenómenos. Ese trabajo incluye una breve lista de los tipos de ambigüedades –léxica, sintáctica, semántica, pragmática, de ingeniería del software y de error de lenguaje–, y apunta al estudio inicialmente de potenciales oraciones ambiguas en requisitos de software. Una colección de medidas de ambigüedad son aplicadas por un pequeño sistema, llamado LOLITA, introducido en (Kiyavitskaya, Zeni, Mich, and Berry 2007). Este sistema toma en cuenta todas las interpretaciones sintácticamente correctas, luego si encuentra más de un árbol de parseo para una oración, la oración es realmente ambigua. En un experimento con muchos usos del sistema sobre muchos dominios, los resultados indican que apenas un 20 % de las oraciones tienen un sólo árbol de parseo, y que el 47 % tiene 10 o más árboles.

El trabajo de (Tjong 2008) intenta producir reglas de guiado para escribir especificaciones de requerimientos sin ambigüedades. Con una descripción mucho más detallada de posibles clasificaciones de las ambigüedades, su análisis identifica con cierta sistematicidad muchos patrones ambiguos, agrupándolos por conjunción, disyunción, cuantificación y demás. Su propósito es sugerir el reemplazo de un patrón de uso de lenguaje por otro menos ambiguo. Aquí se observa una intención de proveer una solución al humano expresada a nivel de lenguaje humano.

2.4. Cucumber

Por último, cabe destacar a Cucumber [1], un sistema que permite transformar una especificación en código Ruby. De nuevo, la especificación no está en lenguaje natural

[1] http://cukes.info/

sino en un formato de plantilla más intuitiva pero provista de similar rigidez. Una característica importante es el caracter interactivo e iterativo del sistema para ir corrigiendo en sucesivos pasos el código obtenido. De todas formas, para especificaciones de sistemas más grandes, el enfoque parece poco extensible. Y claramente, es dependiente del lenguaje de destino.

2.5. Características Relevadas

Después de haber analizado estos sistemas, identificamos algunas características importantes para suministrar a nuestro sistema.

- Flexibilidad en el formato de entrada de especificación en lenguaje natural, lo cual permite que nuestro servicio sea accesible a personas no capacitadas en informática y/o formalización de requerimientos.
- Una representación objetivo que sea de propósito más general, como la de un sistema de transiciones. Concretamente, un grafo etiquetado que representa las secuencias de acciones expresadas en el texto.
- Tratamiento en jerarquía o generalización de fenómenos o patrones.
- Enfoque que permita cierta interacción y retroalimentación desde el humano. En particular, representaciones más intuitivas para comunicar los resultados del análisis.
- Independencia de un lenguaje particular de codificación o de representación.
- Mayor estudio y desarrollo para el idioma español de los fenómenos observados y de las heurísticas y decisiones para su interpretación.

Capítulo 3

Sistemas *Human-in-the-loop*

(Volver al Índice general)

Como se expone en los capítulos anteriores, nuestro sistema propone resolver la representación formal de una especificación de requerimientos funcionales de software a través de un enfoque que no establezca pesados formatos a un analista o usuario encargado de la confección de un caso de uso. Formatos que exigen completar rígidos formularios, abundantes en conceptos ajenos al humano no educado en áreas de análisis de requerimientos. Por el contrario, pretendemos mantener el formato de especificación en lenguaje natural, lo más cercano al nivel de comprensión humana general, y dar tratamiento a esta expresión para capturar secuencias de acciones deseadas en la interacción entre un agente y un sistema. Una propuesta de estandarización para un formato liviano se introduce en la subsección 5.1.1. A la vez, se quiere ayudar a quien especifica a identificar otras características presentes en su texto, tal como los fenómenos semánticos que luego se presentan en este trabajo.

3.1. Un Cuerpo de Casos de Ejemplo

Para entender cómo analizar casos de uso, es preciso hacerse de un cuerpo de casos de ejemplo, preferentemente anotados con algunas categorías de interés, para identificar los múltiples fenómenos que se intentan tratar. A la hora de proveernos de ejemplos de casos de uso, se consultó a un experto en el dominio de corpus respecto a la existencia y disponibilidad de un corpus semejante, y se obtuvo una respuesta negativa. Análogamente, la literatura relacionada no hace mención a estos recursos. Luego de una profunda inspección de recursos digitales mediante los más conocidos motores de búsqueda web, se observa que:

- no es posible proveerse sino de un muy **reducido corpus de documentos de especificación** de requerimientos de software;

- tal corpus no existe como un recurso así identificado y hecho disponible, ni siquiera como conjunto de documentos, sino que debió ser construido a partir de **agregar documentos encontrados de manera individual**;
- el volumen del corpus en construcción **se va reduciendo aún más a medida que se agregan restricciones de filtrado** a la información contenida en los documentos, por ejemplo: cuando exigimos que el documento presente una sección de casos de uso, cuando exigimos que la sección de casos de uso no sea de diagramas sino de textos, cuando exigimos que el documento esté en español, etc;
- si bien existen estándares o convenciones para la confección de casos de uso, cada documento encontrado presenta un estándar particular, con más o menos categorías de datos, presentado en un **formato tabular o textual muy dependiente del documento**. Por lo tanto se vuelve impracticable la extracción automática del texto de interés y se requiere entonces una extracción a mano;
- al no disponerse de un corpus obtenido como un conjunto de casos de uso así identificado, en particular **no se dispone de un corpus anotado**, en el sentido de ejemplos anotados por ciertas categorías de interés para constituir entradas de entrenamiento para mecanismos de aprendizaje automático.

3.2. Representaciones en Grafos Decorados

Como una característica de relevancia, se desea obtener una representación de utilidad general, a diferencia de las obtenidas en los trabajos relacionados que eran muy específicas o dependientes del sistema en particular. Se propone una representación en grafos con arcos decorados por acciones. Esta representación subsume toda otra, y se utiliza en variadas áreas de la computación, dándole así a nuestro sistema su caracter de representación general.

Además, se pretende obtener información asociada a la especificación tal que se exprese a nivel de consumo humano y junto a la representación en grafo, capturen intuitivamente la secuencia de acciones que se especifican. Por lo tanto, es importante que el conocimiento que se obtenga de analizar un caso de uso pueda ser comprendido por el humano. De esta forma, el grafo de representación del caso de uso que se desea obtener tiene la estructura de un sistema de transiciones etiquetado por acciones o *predicados* identificados a partir de la secuencia que el caso de uso expresa. A su vez, un eventual análisis posterior y externo al sistema puede transformar este grafo en otro, que contenga posiblemente parte de las acciones del primero. Luego, los predicados que el grafo contiene como etiquetas de los arcos deben tener asociados mecanismos para

recuperar una representación fiel a la original de los mismos: acciones y actores en lenguaje natural que expresen las transiciones de este nuevo grafo.

3.3. Sistemas *Human-in-the-loop*

Frente a este escenario, las decisiones de diseño del sistema pasan a intentar resolver dos cuestiones fundamentales: cómo obtener más información, y cómo procesar de la mejor manera un volumen reducido de información no anotada. La estrategia escogida para dar tratamiento a ambas es la de diseñar al sistema como uno de características ***Human-in-the-loop***.

Un modelo *Human-in-the-loop* es definido como uno que requiere interacción humana. Se asocia con sistemas en los que predomina un enfoque constructivo, dinámico, en donde el humano tiene participación fundamental en algún punto del proceso. Es tan importante la interacción que permite identificar problemas, requerimientos, decisiones, que de otra forma serían muy difíciles de conseguir. Parece entonces muy apropiado a un sistema que da tratamiento a un análisis tan constructivo como lo es la elicitación de requerimientos. Sumado a lo recién mencionado, una enorme ventaja que se desprende es que el humano puede controlar la salida de un evento o proceso, aceptando o modificando lo obtenido por el sistema. De esta forma, el sistema mejora con esta información clave sobre su rendimiento y el humano está entrenándose en sus habilidades de reconocer ciertos fenómenos al estar inmerso en el evento o proceso.

En el contexto industrial, en los últimos 50 años, una clara **división de responsabilidades entre el humano y la máquina** ha evolucionado en base a la habilidad óptima de cada uno (Pretlove and Skourup 2007). Esto lleva al humano a ocuparse de un conjunto de tareas reducido pero crucial como supervisión, control de excepciones, optimización y mantenimiento. Este enfoque, aplicado al contexto del desarrollo de software, nos permite pensar en un humano que realiza la tarea crucial de reconocer y sugerir soluciones a los fenómenos presentes en las expresiones en lenguaje natural con las que se especifican requerimientos en un caso de uso.

El problema de procesar de la mejor manera un volumen reducido de información no anotada se intenta resolver justamente complementando la información que provee la interacción y el modelo de decisión diseñado a partir de la inspección del cuerpo de casos de ejemplos.

La estrategia de un humano anotador es ya conocida y utilizada en varias **aplicaciones exitosas**. Quizá una de las de mayor actualidad sea la del *Amazon Mechanical Turk* [1]. Este servicio web permite a individuos o entidades denominadas *requesters*

[1] http://www.mturk.com/

solicitar *Tareas de Inteligencia Humana* que las computadoras aún no son capaces de resolver, como por ejemplo elegir la mejor fotografía de entre un conjunto, o identificar un artista en una canción. Los trabajadores, también denominados *Providers* o *Turkers*, pueden elegir y realizar una tarea a cambio de un pago previamente fijado por el *requester*.

Estas propuestas se enmarcan en la estrategia general de ***crowdsourcing***, entendida como la práctica de obtener servicios, ideas o contenidos solicitando la contribución de una gran cantidad de personas, casi exclusivamente por medios digitales, generalmente la comunidad online. Uno de los principales objetivos en obtener estos grandes volúmenes de datos es la aplicación de aprendizaje automático sobre cuerpos de ejemplos anotados. Como bien es notado en (Wang, Hoang, and Kan 2010), el enfoque tradicional de expertos anotadores para obtener información de calidad se vio poco a poco reemplazado por otro que afirma que la redundancia en grandes cantidades de ejemplos anotados actúa como filtro de los datos ruidosos o dispersos. Esta hipótesis de cantidad y redundancia es claramente fundamental en el enfoque. En ese mismo trabajo se analizan las aplicaciones más importantes del *crowdsourcing* particularmente para datos útiles a las tareas de Procesamiento de Lenguaje Natural –PLN–, enfatizando que los trabajadores de la anotación no deben ser expertos. Además, destaca que en general las tareas de anotación fueron diseñadas como entretenimiento para realizarse en el curso de sesiones cortas, con el fin de mantener al trabajador activo, pero que en una aplicación como el *Amazon Mechanical Turk*, la motivación es el beneficio económico.

Es importante entonces diseñar tareas de anotación que resulten sencillas de realizar para el anotador, que eviten la dispersión de la información provista debido a ambigüedades o complejidades en la tarea, y que el trabajador se sienta motivado para mantenerse productivo. También sería ideal que, mediante las características de atractiva y simple de la tarea, se pudiera mantener honesto en sus respuestas para una mayor calidad de los datos. En nuestro sistema, la honestidad del humano se asegura dado que su respuesta lo beneficia en el análisis de sus requerimientos.

En el trabajo de (Munro, Bethard, Kuperman, Lai, Melnick, Potts, Schnoebelen, and Tily 2010) se recopilan proyectos recientes que utilizan información interactiva para estudios y evaluación de tareas de lenguaje, con performance en algunos casos superior a la de experimentos controlados de laboratorio. Tareas como las de transparencia de frases verbales, predictibilidad contextual o análisis de frecuencia metafórica están aplicando este enfoque con éxito, demostrando que el contexto original del PLN se vuelve más amplio incluyendo disciplinas desde la semántica a la psicolingüística.

3.4. La Especificación en Lenguaje Natural

Una de las estrategias más usadas para especificar un clase particular de requisitos, llamados funcionales –debido a que refieren a una funcionalidad del sistema en cierto escenario de interacción–, es la definición de un conjunto de **casos de uso** como parte del documento de especificación de requerimientos de software. Un caso de uso es una descripción de una secuencia de pasos precisos, claros y realizables que representa una interacción entre un usuario y el sistema para alcanzar cierto objetivo. Un caso de uso constituye una interfaz natural y amigable para describir cómo se espera que un sistema se comporte al interactuar con un usuario, ya que presenta una curva de aprendizaje prácticamente nula –a lo sumo, exigiendo un vocabulario controlado. Así, permite el acceso de un gran rango de roles dentro de una organización de desarrollo de software, incluso, y en particular, roles externos como clientes.

Aunque se lo suele confundir con otra técnica más visual de diagrama de caso de uso, como bien se expresa en (Larman 2004), un caso de uso no es un diagrama, es un texto, y el modelado de casos de uso es un acto de escritura de textos. El siguiente es un ejemplo de caso de uso para el escenario de la desactivación de la alarma de un sistema.

Precondiciones:

$$alarma\_activada() \wedge hora\_alarma == hora\_actual$$

Flujo normal:

1 El sistema emite un sonido y muestra un cartel de alarma.
2 El sistema muestra un menú de opciones: "Desactivar", "Posponer" .
3 El usuario selecciona la opción "Desactivar".
4 El sistema deja de emitir el sonido.

Estos casos de uso, si bien escapan de estándares rigurosos, comparten en general la presencia de una sección de flujo normal expresado en lenguaje natural. Es ésa sección la que utilizaremos como entrada a interpretar.

3.5. Un Sistema Interactivo

Resulta evidente que el diseño de nuestro sistema no debe dirigirse hacia unidades de información sólo tratables automáticamente, lo cual posiblemente llevaría a una representación:

- demasiado simple, como por ejemplo, la que se obtendría si no se preserva infor-

mación de la expresión textual ingresada originalmente en el caso de uso, que debe transformarse en otra análoga basada en aquélla para comprensión del humano que recibe los resultados de la aplicación de nuestro sistema; y/o

- demasiado compleja, si por ejemplo se intenta aplicar un algoritmo de desambiguación de sentidos.

En el capitulo siguiente de Arquitectura del Sistema se desarrollarán los detalles en las decisiones de diseño para efectivamente utilizar el enfoque *Human-in-the-loop*. En definitiva, para un sistema que intenta obtener una representación de conocimiento muy general, a nivel de consumo humano y que se obtiene de la interpretación de un conjunto muy variado y complejo de expresiones y estructuras ambiguas:

- se desarrollan estrategias de detección y recuperación de información a partir del pequeño cuerpo inicial de ejemplos de casos de uso;
- se obtienen representaciones de la especificación, sobre las cuales pueden realizarse transformaciones posteriores, externas a nuestro sistema;
- se preserva relación con el texto original de entrada para su recuperación y devolución al humano tras esas eventuales transformaciones posteriores;
- no se hace desambiguación de sentidos de palabras, ya que no se pretende mapear mediante función el conocimiento a una estructura como la de una ontología;
- las ambigüedades imprescindibles a resolver para capturar los predicados involucrados y sus relaciones son inherentes a la flexibilidad de la expresión en lenguaje natural del caso de uso. Y si estas ambigüedades escapan del modelo de decisión implementado, se resuelven interactuando con el humano.

Es importante mencionar que el sistema fue diseñado para retroalimentarse, lo que significa que toda interacción resulte en una anotación –en el sentido de la construcción de un conjunto de ejemplos para aplicar métodos de aprendizaje automático. En una instancia futura en la que se disponga de un gran volumen de ejemplos, desde estas anotaciones podrían inferirse qué interpretaciones para expresiones ambiguas son más probables o apropiadas de ofrecer al humano y facilitar la resolución de ambigüedades. Esto, como se expone más detalladamente en 5.3.1, implica discernir apropiadamente acerca de las diferentes partes de la información que deben almacenarse como conocimiento adquirido.

Capítulo 4

Arquitectura General del Sistema

(Volver al Índice general)

La arquitectura de alto nivel propuesta para el sistema se describe a continuación. Una representación gráfica muy útil aparece en la Figura 4.1.

Esta arquitectura se caracteriza por un flujo de datos a través de una secuencia de etapas claramente definidas. A partir de un texto plano en lenguaje natural que contiene una especificación de requerimiento funcional de software expresada mediante un caso de uso, se aplican las siguientes componentes:

Preprocesamiento del caso de uso

Se reconocen estructuras morfosintácticas en el texto plano de entrada;

Extracción de elementos relevantes

Se extraen unidades de información desde las estructuras preprocesadas, posiblemente manipulando elementos de conocimiento estático para resolver situaciones de ambigüedad;

Interpretación de elementos extraidos para identificar estructuras de predicados

Se aplican listas de reglas de decisión sobre las unidades de información, interpretando así varios fenómenos semánticos y sintácticos necesarios para construir el grafo;

Construcción de grafo a partir de la información interpretada

Se obtiene un sistema de transiciones etiquetadas (STE), representado en un formato de texto plano –en adelante nos referiremos indistintamente a esta estructura como grafo o STE.

Además, un módulo que interactúa con partes de la secuencia descrita en puntos determinantes:

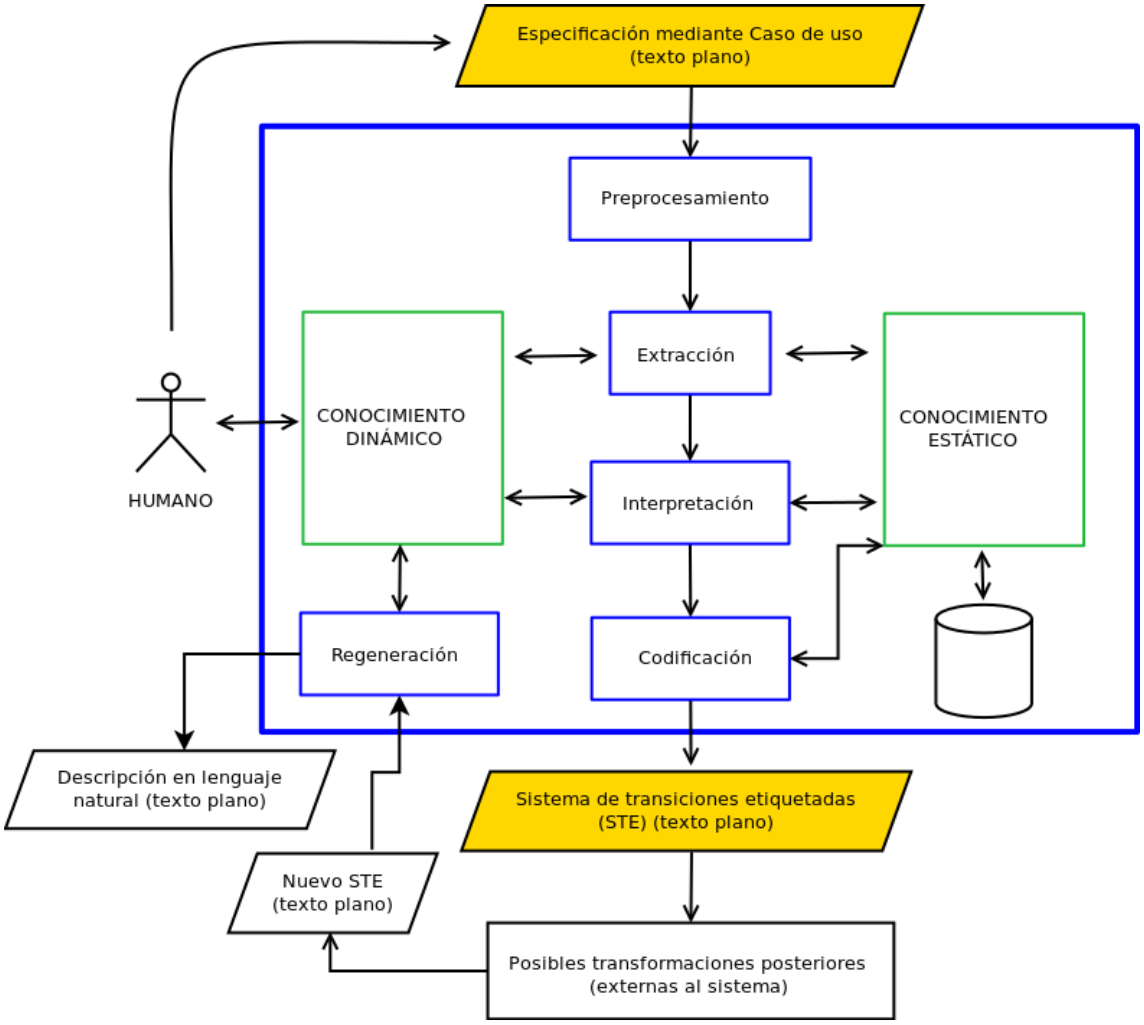


Figura 4.1: Arquitectura de alto nivel del sistema

Regeneración de Lenguaje Natural

> Se regeneran segmentos significativos de la expresión original del texto de entrada en lenguaje natural, necesarios para complementar los procesos de Extracción y de Interpretación.

En la arquitectura de alto nivel existen también un par de componentes de Conocimiento. Son éstos los que proveen la información y los mecanismos de decisión para que los diferentes módulos puedan efectuar sus procesos. Con este conocimiento, por ejemplo, se pueden detectar y extraer elementos característicos de ambigüedades y luego tratar de interpretar estos fenómenos de la forma correcta. A través de las siguientes secciones presentaremos decisiones de diseño de mayor detalle en los módulos que conforman el sistema, sus objetivos y tareas, y las estructuras de datos que utilizan. En particular, explicaremos las estrategias que proponemos para la adquisición y utilización de conocimiento.

4.1. Preprocesamiento

(Volver al Índice general)

Como ya se mencionó, la característica principal en la motivación y diseño de nuestro sistema es la enorme flexibilidad que se permite al momento de expresar requerimientos funcionales a través de casos de uso. El tipo de dato de entrada es un texto plano en lenguaje natural con un formato muy relajado, por lo tanto se está abierto al ingreso de texto de dominio variado, muy posiblemente con presencia de, por ejemplo, vocabulario de dominio, estructuras sintácticas complejas, y sobre todo, ambigüedades, es decir, sentidos implícitos en formas particulares de expresarse para los que no hay una única interpretación. Más aún, el texto en lenguaje natural puede presentar errores gramaticales, ortográficos, e incluso elementos inexistentes en un idioma, ingresados maliciosamente. Así, frente a este extensísimo dominio de texto de partida, característico de nuestro sistema, debemos responder con robustez.

La etapa de Preprocesamiento es justamente la primera en proveer parte del tratamiento robusto; a la vez, constituye un proceso de convergencia o de simplificación. El módulo de Preprocesamiento transforma un texto plano –que contiene un caso de uso expresado en lenguaje natural– en una estructura de árbol de dependencias sintácticas que simplifica el reconocimiento y tratamiento de los fenómenos presentes en el lenguaje. A la vez provee parte de esa robustez convirtiendo el texto de extenso dominio en elementos morfosintácticos bien conocidos. Para esa transformación se utiliza la herramienta FreeLing 3.0.

FreeLing es un conjunto de herramientas orientadas al desarrollador, que proveen varios servicios de análisis de lenguaje –ver (Padró and Stanilovsky 2012), (Padró 2011), (Lluís Padró and Castellón 2010), Atserias et al. (2006), Carreras et al. (2004). En particular, nuestro sistema invoca el servicio de parseo de dependencias (*dependency parsing*), que incluye además: tokenización, partición por oraciones, análisis morfológico, lematización y anotación morfosintáctica (*part-of-speech tagging*). El Preprocesamiento a nivel de *dependency parsing* es fundamental ya que la jerarquía de dependencias captura características muy importantes para nuestro sistema, entre otras, y como se desarrollará luego, el alcance de un verbo.

Ya que trata con estructuras sintácticas más complejas, el *dependency parsing* produce un árbol con posiblemente más error que los obtenidos mediante otros parseos –por ejemplo el *shallow parsing*. Como un ejemplo simple de este error, consideremos la oración *"Compró un tapado de piel de foca"*. [1] El árbol de *dependency parsing* que se obtiene con FreeLing es el de la Figura 4.2. El sintagma proposicional *de foca* es

[1] Ejemplo clásico aunque desafortunado. El autor adhiere a sus virtudes sintácticas y no a las semánticas. (Volver al texto)

```
grup-verb/top/(compró comprar VMIS3S0 -) [
  sn/dobj/(tapado tapar VMP00SM -) [
    indef-ms/espec/(un uno DI0MS0 -)
    sp-de/sp-mod/(de de SPS00 -) [
      sn/obj-prep/(piel piel NCFS000 -)
    ]
    sp-de/sp-mod/(de de SPS00 -) [
      sn/obj-prep/(foca foca NCFS000 -)
    ]
  ]
  F-term/term/(. . Fp -)
]
```

Figura 4.2: Representación en texto plano de un árbol de *dependency parsing*

analizado como adjunto o modificador del núcleo *tapado* del objeto directo, cuando en realidad es un adjunto del sustantivo *piel*.

La situación del ejemplo es esencialmente un problema de ambigüedad, y muestra las dificultades que debemos resolver respecto a otros casos ambiguos que sí nos son relevantes, como el de la identificación de estructuras coordinantes conjuntivas o disyuntivas. Más adelante, se explicará cómo nuestro sistema puede contribuir a la adquisición léxica de un mayor volumen de recursos para resolver éstas y otras ambigüedades.

El análisis que realiza FreeLing es costoso en tiempo de cómputo, del orden de magnitud de segundos, por lo tanto domina la complejidad de todo el sistema y es así que no hicimos más restricciones de diseño en la implementacion respecto a la eficiencia. Este coste de cómputo es una situacion general del preproceso linguístico.

Como parte del tratamiento inicial del caso de uso en lenguaje natural, y con la finalidad de recuperar segmentos significativos del texto original, la etapa de Preprocesamiento realiza previo al parseo recién explicado, una tarea de extracción de segmentos significativos del texto original que se conservan para un eventual uso por parte de la componente de regeneración de lenguaje natural.

Nuestro módulo de Preprocesamiento es entonces una abstracción del parseo de dependencias, que nos permite, por ejemplo, cambiar en la implementación la herramienta de análisis que se usa, o bien parametrizar por idioma y decidir qué herramienta utilizar y qué pasos previos al parseo se requieran.

4.2. El Concepto de Predicado

(Volver al Índice general)

Para entender los objetivos y los procesos que caracterizan a las demás componentes del sistema y las relaciones que existen entre las mismas, es oportuno introducir el concepto de predicado, central a nuestro estudio y que estará muy presente de aquí en adelante en este trabajo.

Un predicado es una unidad de información que se corresponde con una acción simple en la secuencia de interacciones humano-sistema que se especifica en un caso de uso. Por ejemplo, en la secuencia

$$\begin{aligned} &1\ )\ \textit{El usuario inserta una moneda de un peso.} \\ &2\ )\ \textit{El sistema retorna un café y emite un sonido.} \end{aligned} \tag{4.1}$$

se pretende idealmente considerar predicados a las unidades *"El usuario inserta una moneda de un peso"*, *"El sistema retorna un café"* y *"El sistema emite un sonido"*, ya que el paso *2* presenta dos acciones simples coordinadas en conjunción.

De esta manera, esta definición es muy similar al concepto sintáctico de predicado: una unidad mínima de sentido, sin autonomía sintáctica. La unidad por defecto que lo conforma es el verbo conjugado, y cada verbo con sus dependientes –argumentos o adjuntos– forman un predicado. Son éstos los principales elementos de información que deseamos reconocer y extraer de las expresiones en lenguaje natural, para luego interpretarlos apropiadamente como predicados.

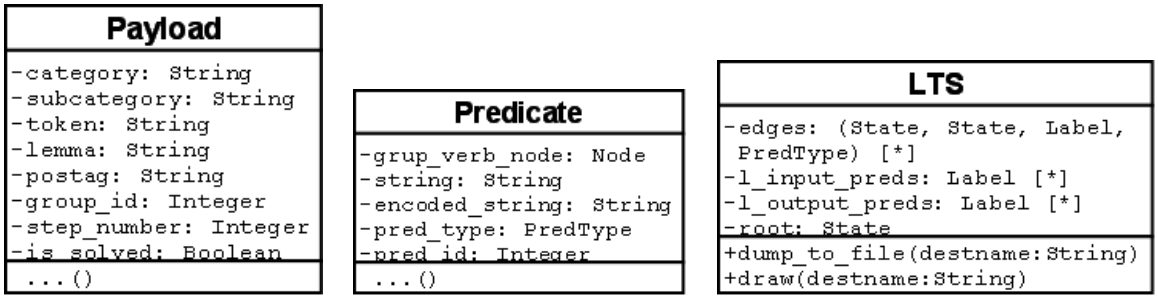
Luego, nos interesan sólo sintagmas verbales de verbos conjugados y sus adjuntos, es decir, palabras que se presentan junto al verbo no sólo en proximidad textual sino en dependencia sintáctica. Este patrón simple dominado por un verbo conjugado es buscado sobre la estructura arbórea del resultado de la etapa de Preprocesamiento. Exponemos más detalles de esto en la siguiente sección.

4.3. Extracción

(Volver al Índice general)

A partir del árbol de dependencias obtenido por el módulo de Preprocesamiento, queremos reconocer a la vez los predicados presentes en la especificación y las relaciones entre los mismos. De esta forma, cada predicado será una etiqueta que decora un arco del grafo objetivo, y la estructura de relaciones entre predicados implicará la definición de nodos y arcos del grafo.

Como se ve en la Figura 4.2, el árbol de dependencias que provee FreeLing está



(a) `Payload` (b) `Predicate` (c) `LTS`

Figura 4.3: Clases propuestas para las estructuras internas de información

representado mediante un texto plano con símbolos y cadenas especiales que delimitan la jerarquía de dependencias que la herramienta pudo analizar. Resulta a la vez más sistemático y simple buscar información en una estructura de datos que represente a un árbol, en lugar de usar complejas expresiones regulares sobre este texto plano de dependencias. Esta necesidad de disponer de una estructura de datos de árbol se vuelve más notoria si se considera que eventualmente se deberán hacer modificaciones a la jerarquía para, entre otras tareas, extraer, reorganizar y/o eliminar subárboles.

Esta tarea de transformación de una representación del árbol a otra de mayor nivel de abstracción no es suficiente para que lo consideremos como el procesamiento central que define por sí mismo a un módulo. Más adelante, cuando se exponga la propuesta de incluir en esta etapa una estrategia de detección temprana de la ambigüedad, será claro que esta transformación de representaciones de árboles implica una tarea de optimización al extraer claves textuales mientras se construye la nueva estructura de árbol.

Haciendo uso de una estructura de datos de árbol `Tree`, el resultado del Preprocesamiento de texto plano es parseado para obtener representaciones de más alto nivel de abstracción. Cada árbol de nivel 0 de indentación en la secuencia de árboles de *dependency parsing* –aquellos identificados con el no terminal `top` en su primera línea– se convierte en un elemento de tipo `Tree`. De aquí en más, es esta estructura de datos a la que nos referiremos como *árbol*. Esta secuencia ordenada de árboles con información extraídas y estructurada es el dato entregado por el módulo de extracción a la siguiente etapa.

Los elementos básicos a detectar y extraer son los verbos conjugados y sus dependientes. Esto nos permitirá delimitar cada predicado. A la vez, debemos extraer información sobre la relación entre estos predicados, ya que como se vio en el ejemplo 4.1, una simple expresión en lenguaje natural puede contener varios predicados y jerarquizados entre sí de formas que, aunque no sean complejas de entender y de representar una vez extraídass, sí que pueden ser complejas de identificar.

Cada nodo del árbol contiene gran parte de esta información que se desea extraer, almacenada en una instancia de la clase `Payload`. Tal como se puede observar en la Figura 4.3(a), esta clase que se propone representa, entre sus principales atributos:

- con `category` y `subcategory` los respectivos primer y segundo no terminales que se analizaron para una palabra, presentes al principio de cada línea del resultado de la etapa de Preprocesamiento, formateados de la forma `<category>/<subcategory>/...` –por ejemplo, `sn` y `dobj` en la segunda línea de la Figura 4.2;

- con `token`, `lemma` y `postag`, respectivamente, el token –o lexema–, lema y etiqueta de POS-tagging para esa palabra –por ejemplo, `tapado`, `tapar` y `VMP00SM` en la segunda línea de la Figura 4.2;

- con `grup_id` el sintagma verbal al que pertenece esa palabra en la jerarquía de dependencias. Así, el parseo para obtener el árbol a partir del texto plano resultante del Preprocesamiento no sólo construye cada nodo para cada palabra reconociendo los atributos antes mencionados, sino que es un poco más especializado y marca cada verbo núcleo de un sintagma verbal con un identificador entero único. Los identificadores están relacionados entre sí de forma que permiten, además de representar unívocamente cada sintagma verbal –y luego cada predicado–, preservar la relación de orden textual entre ellos. Este elemento provee eficiencia a la inspección de dependientes en una búsqueda sobre el árbol ya que puede consultarse en orden constante el sintagma verbal al que pertenece cada uno de estos argumentos o adjuntos sin tener que volver hacia atrás en la jerarquía hasta el ancestro verbo más cercano. Debe notarse también que por diseño de la herramienta usada durante el Preprocesamiento, el sujeto correspondiente a un sintagma verbal aparece como uno de los dependientes del verbo.

Disponemos entonces de mecanismos directos para detectar cada nodo de verbo conjugado en el árbol –a través de métodos accesores a los atributos `category` y `postag`–, y también para detectar los dependientes de cada verbo –usando `group_id`. Estos mecanismos, combinados con las funcionalidades sobre árboles que provee la interfaz de la estructura de datos `Tree`, son de utilidad central en la extracción de las unidades de información necesarias para, en la etapa de interpretación, determinar los predicados presentes en la especificación original. Las formas textual y lematizada de cada palabra –accediendo, respectivamente, a `token` y `lemma`– se usan también para resolver otros fenómenos que se proponen estudiar en este trabajo, como algunos de los desarrollados en la sección 5.1.

4.4. Interpretación

(Volver al Índice general)

Como bien lo expresa el título de este trabajo, nuestro objetivo es interpretar especificaciones de software. Más específicamente, y de acuerdo con los conceptos que establecimos, el sistema propuesto transforma un caso de uso en un grafo construido según la secuencia de acciones especificadas. Esto es, identifica los predicados presentes en el texto en lenguaje natural y los representa en una estructura más apropiada para su manipulación y estudio computacionales. Interpreta la especificación a partir de la interpretación de las unidades más simples de representación de acciones y de la estructura que las relaciona.

El módulo de Interpretación es, junto con el de Extracción, el núcleo de nuestro sistema ya que es el que obtiene cada uno de los predicados, así como las relaciones entre ellos, a partir de la información clave detectada y extraídas en la etapa previa. La tarea de interpretación es esencialmente, por tanto, la de resolver las ambigüedades con las que se estructuran en lenguaje natural las relaciones entre los predicados, haciendo uso de mecanismos de decisión propios del sistema, aplicados sobre diferentes elementos de esa especificación.

Para disponer de tales mecanismos, hemos diseñado estrategias que construyan reglas de decisión de manera que si un elemento satisface la condición de aplicabilidad de una regla, entonces la misma es aplicada sobre una estructura para la que ese elemento es significativo. Los efectos de una decisión como ésta intentan resolver de la manera más precisa tanto la identificación de un predicado como la jerarquía o relación entre los diferentes predicados. Asimismo, como parte de esas estrategias, debemos estudiar, entre otras, qué elementos serán más apropiados para usarlos como claves de decisión, y qué criterios se consideren para establecer una jerarquía de precedencias entre las reglas. Además de esta interpretación de predicados, este módulo se encarga de resolver otros fenómenos expuestos en la sección 5.1.

La lista de datos de tipo `Tree` que se obtiene desde la etapa de Extracción es tratada por el módulo de Interpretación, a fin de obtener la estructura de relaciones de predicados presentes en la especificación. Esta estructura es un grafo (como un dato G de un tipo `Graph`) con arcos decorados por predicados, que construimos como instancias de una clase `Predicate` definida como en la Figura 4.3(b).

- Según lo desarrollado en la sección 4.2, dado que el verbo conjugado es la unidad constitutiva de un predicado, el atributo `grup_verb_node` guarda una referencia al nodo del árbol donde se encuentra ese verbo. Esta referencia es útil para obtener no sólo esta unidad sino también los dependientes del verbo, es decir el resto de

los elementos posibles que acompañan al verbo conjugado como significativas componentes del predicado. Estos elementos se seleccionan justamente de entre todos los tokens que componen ese grupo verbal.

- Cuando todos los elementos textuales significativos para ese predicado han sido seleccionados, se desea con ellos construir una cadena o segmento de palabras que exprese con la mayor fidelidad posible lo que originalmente expresaba el caso de uso en el fragmento de texto correspondiente a este predicado. El atributo `string` puede contener simplemente una concatenación, en algún orden, de los tokens significativos –lo cual sea muy posiblemente poco fiel al original–, o bien una cadena como la deseada que se obtiene por interacción con el módulo de Regeneración de Lenguaje Natural.

- `encoded_string` es una forma codificada de la cadena de `string`; su objetivo es explicado en la sección 4.5.

- El entero `pred_id` identifica únivocamente a cada predicado. Será utilizado para la construcción de `encoded_string`. También para contribuir en las heurísticas de decisión, ya que el valor que toma este atributo entero es definido como el incremento en 1 del valor anterior a medida que se recorre la lista de árboles de `Payload`, luego se relaciona con el orden textual con el que los predicados "aparecen" en el caso de uso –en realidad, los que aparecen son los tokens significativos que se seleccionaron de ese grupo verbal.

- el atributo `pred_type` guarda información sobre el fenómeno de tipo de predicado, a explicar en la sección 5.1.

Diversas claves, como expresiones de interés formadas por una o más palabras, formas textuales de la disposición de las palabras en las oraciones, presencia o ausencia de ciertas marcas o nexos, entre otros, son los criterios generales para construir reglas muy simples que resuelvan las ambigüedades con las que se expresan las relaciones entre predicados. Inicialmente capturando algunos pocos casos, para luego ser generalizadas progresivamente a partir de la unificación o simplificación de claves de decisión. A medida que se van interpretando estas claves, los predicados ya obtenidos se agregan apropiadamente al grafo G en construcción. G es finalmente obtenido como salida del proceso principal de este módulo: la especificación ha sido interpretada y ya se dispone de un grafo que representa su información.

4.5. Codificación del Grafo

(Volver al Índice general)

La etapa de Interpretación construye un grafo de representación de la secuencia de acciones de interacción humano-sistema descrita en la especificación. Es por esto que identificamos a este grafo como un sistema de transiciones etiquetadas –STE. Pero como el grafo obtenido es de utilidad general, consideramos más apropiado una etapa adicional que introduzca abstracción sobre este grafo al ocultar las representaciones textuales de cada predicado que decora un arco. Así, definimos una codificación del grafo para generar un STE que preserve la estructura de relaciones entre los predicados pero que se abstraiga de los segmentos de textos con el que fácilmente identificamos cada predicado.

Además, nuestro grafo codificado *G'* puede llevar información obtenida del grafo base *G* adicional a la de estructura de nodos y arcos. Así, con un mínimo coste de espacio en la codificación, la parte que haga uso de *G'* puede disponer eficientemente de tales datos adicionales que de otra forma le sería costoso de obtener –por ejemplo, nuestra codificación podría incluir información que venga siendo recolectada durante la construcción de *G*. Tal STE *G'* será el que finalmente devuelva nuestro sistema como dato de salida.

Luego, si se desea hacer transformaciones o análisis sobre este STE, se estará trabajando con una versión más liviana y abstracta del grafo: sólo etiquetas genéricas que nada informan sobre las expresiones textuales que lo generaron. Esta codificación, además de una etapa de simplificación para las aplicaciones externas que usen el resultado *G'*, puede significar, como toda buena abstracción, un mecanismo de seguridad si se desea mantener las estructuras textuales de los predicados fuera del conocimiento de agentes externos a nuestro sistema y que los mismos dispongan como interfaz a esta representación codificada.

Y en el caso en que los análisis aplicados sobre el STE –o transformaciones que generen nuevos STE– requieran más expresividad, podemos puede recuperar con alta fidelidad las expresiones del texto original a través del módulo de Regeneración de Lenguaje Natural. Si los atibutos `string` de las instancias de `Predicate` ya estaban definidos como estas expresiones regeneradas, entonces bastará con hacer simplemente uso de las asociaciones entre `string` y `encoded_string`. La Figura 4.4 muestra el archivo JSON del grafo codificado para el siguiente caso de uso:

El usuario introduce una moneda.
La máquina retorna un té.

```
{
    "root": "s1",
    "transitions": {
        "s1": {
            "s2": {
                "label": "label_1"
            }
        },
        "s2": {
            "s3": {
                "label": "label_2"
            }
        },
        "s3": {}
    }
}
```

Figura 4.4: Ejemplo de un archivo JSON para grafo codificado

El grafo original se codifica creando una instancia de una clase `LTS`, definida con los siguientes atributos:

- `edges`, una lista de arcos o 4-uplas de la forma `(st_1, st_2, label, pred_type)`, donde `st_1, st_2` son nodos obtenidos de la estructura de nodos del grafo original, y `label` es una etiqueta codificada para el predicado que decora ese arco del grafo;
- `l_input_preds` una lista de las etiquetas correspondientes a todos los predicados de tipo *input* presentes en el grafo –esta clasificación de tipos de predicados se explica en la sección 5.1;
- análogamente, una lista de etiquetas de predicados de tipo *output* `l_output_preds`;
- `root`, el nodo raíz del grafo.

Además, definimos algunos métodos como `draw()` –que hace uso de una librería de gráficos para obtener una imagen del grafo– y `dump_to_file()`, que genera un archivo de texto plano en un formato estandar, como salida de nuestro procesamiento. Esta clase `LTS` se presenta en la Figura 4.3(c).

Una codificación muy simple que se puede proponer consiste en construir cada etiqueta codificada de la forma `label_i`, donde `i` es un índice único del predicado, obtenido por acceso al atributo `pred_id` de la correspondiente instancia de `Predicate`.

Este grafo codificado se representa en formato JSON. Fue elegido dado que es sencillo transformar la información que disponemos a este formato, sumado a que es convenido como un estándar para intercambiar datos entre sistemas. La representación

de nuestro grafo consiste en un objeto JSON por cada uno de los 4 atributos de la instancia de `LTS`. Para cada objeto de lista de etiquetas, el valor es un arreglo JSON; para el objeto de nodo raíz, el valor es una cadena JSON. En el caso del objeto para los arcos, el valor es un objeto de objetos anidados.

4.6. Una arquitectura *Human-in-the-Loop*

(Volver al Índice general)

Las componentes de la arquitectura general hasta ahora introducidas realizan tareas de análisis y de transformación de los datos en sucesivas etapas hasta obtener el grafo de representación de la especificación de entrada. Sus heurísticas internas requieren tomar decisiones sobre diferentes fenómenos presentes en las estructuras de datos, para lo cual se deben valer de información concreta que accione estas decisiones.

Por ejemplo, si se desea detectar una expresión que indica cierta ambigüedad en un patrón de texto dado, esa expresión debe ser conocida por nuestro sistema como marca de tal ambigüedad, así como también se debe conocer ese patrón. Más aún, debe poseerse información acerca de cómo extraer ese elemento de marca y cómo resolverlo en el contexto del patrón para una interpretación apropiada del fenómeno en cuestión.

4.6.1. Las componentes de Conocimiento

(Volver al Índice general)

La arquitectura que proponemos presenta dos tipos de conocimiento. La clasificación de estas componentes en Conocimiento Estático y Dinámico expresa el momento en el que el sistema dispone de tal conocimiento, similar a considerar para ciertos módulos si parte de su información se conoce o no exclusivamente en tiempo de ejecución. En definitiva, esto concierne a la fuente o el origen del conocimiento.

La componente de **Conocimiento Estático** consiste en información almacenada en el sistema, inicialmente de poco volumen y obtenida a mano por inspección. Idealmente, es información estática por estable y persistente como complemento fundamental a las heurísticas de extracción e interpretación, ya que si este conocimiento se ha incorporado efectivamente al sistema, se debe a que es considerado representante con alta fidelidad de determinadas claves de resolución de fenómenos.

El Conocimiento Estático queda establecido principalmente durante la implementación del sistema como base inicial de estructuras de datos. Así, una parte importante de este conocimiento está disponible antes de la ejecución sobre un caso de uso de entrada, y su contenido puede modificarse con conocimiento aprendido durante el análisis de una nueva especificación. Este resultado del aprendizaje será persistente en el sistema

hasta la siguiente ejecución de interpretación, la cual eventualmente lo modifique con nuevo aprendizaje.

En cambio, la denominación de un **Conocimiento Dinámico** refiere a datos que el sistema puede aprender sólo durante su ejecución sobre una especificación en lenguaje natural, es decir es dependiente de los datos particulares que se están interpretando. Además, y fundamentalmente, dependen de la información adicional provista por el humano que responde a sugerencias y preguntas del sistema; confieren así la característica central de sistema interactivo.

El Conocimiento Dinámico es una componente fundamental del sistema ya que nuestra arquitectura es de fuerte interacción con el humano. Ambos módulos de Extracción e Interpretación pueden solicitar al humano, mediante mecanismos de interacción de preguntas simples, alguna información adicional que se considere útil para resolver ambigüedades que se encuentren.

Esta interacción es central al sistema y es importante mencionar que involucra muchas aspectos y tareas. La información obtenida dinámicamente desde el humano debe:

- modificar los elementos extraidos y/o interpretados desde el caso de uso, que sean correspondientes a los datos solicitados;
- propagar la resolución de ambigüedades a todas las situaciones análogas posibles, para evitar repreguntar;
- modificar las partes correspondientes del Conocimiento Estático, por ejemplo, retroalimentar las condiciones de aplicabilidad de una regla. De esta manera, lo aprendido es efectivamente almacenado en estructuras de datos y permite incrementar el conocimiento disponible;
- servir para crear hábitos en el humano que mejoren, o al menos simplifiquen, su expresión de casos de uso en lenguaje natural, a partir de que reconozca en la necesidad de interacción la presencia de elementos complejos en el texto.

En el Conocimiento Dinámico también se almacenan las unidades de información extraídass, como se mencionó oportunamente, al inicio del módulo de Preprocesamiento, para un eventual uso por parte de la componente de regeneración de lenguaje natural.

Aunque las denominaciones de los tipos de conocimiento están originalmente inspiradas en diferenciar los mecanismos para aprender –esencialmente, información establecida a mano tras inspección y estudio de fenómenos, o automáticamente en el caso de disponer de conjunto apropiado de ejemplos, versus información dinámicamente provista por el humano en interacción con el sistema–, como hemos explicado,

nuestra clasificación es mucho más abarcativa. La componente de Conocimiento Estático presenta datos tanto identificados y almacenados a mano durante la construcción del sistema, como aprendidos automáticamente a partir de ejecuciones sobre casos de uso de entrada. Asimismo, el Conocimiento Dinámico es en parte automático, como las estructuras para regeneración antes mencionadas, y también obtenido por interacción.

Una estrategia general para integrar estas componentes de conocimiento con los módulos o etapas de procesamiento implica combinar estos conocimientos de forma que se beneficien mutuamente. Frente a un fenómeno a resolver, el sistema consultaría su almacén estático, en busca de claves de decisión. Si las posee, las utiliza. De lo contrario, intenta obtenerlas solicitando ayuda al humano en interacción. Como resultado de ésta, el conocimiento dinámico no sólo brinda información para tratar el fenómeno, sino que también puede aprovecharse para aprender datos que engrosen nuestro conocimiento estático. Claramente, y como se ampliará más adelante en este trabajo, debemos implementar estrategias para incorporar sólo el conocimiento dinámico que se pueda considerar fiable.

4.6.2. La Interacción en Acción

(Volver al Índice general)

Hemos presentado cada módulo de procesamiento involucrado en la secuencia principal desde la especificación de requerimientos de entrada hasta el grafo de representación de salida, así como las componentes de conocimiento que complementan de manera determinante las heurísticas para extraer e interpretar claves textuales de decisión. Toda esta arquitectura hasta ahora introducida parece establecer sólo un sistema de procesamiento lineal de los datos a través de etapas especializadas. Veamos a continuación cómo proponemos redimensionar las estrategias para lograr que el sistema adquiera el enfoque fundamental *Human-in-the-Loop* en toda su potencialidad.

En el diseño propuesto para este sistema, es el módulo de Interpretación el encargado de resolver los fenómenos lingüísticos y semánticos que nos interesan capturar, y que introduciremos en el capítulo siguiente. Aún así, la interpretación de especificaciones en lenguaje natural para obtener grafos de representación es el objetivo del sistema en su totalidad. El módulo de Extracción está, por tanto, dirigido a asistir a la etapa siguiente en su tarea meta. Y claramente es un principio fundamental el tratar de resolver situaciones en el momento más temprano y razonable que sea posible, no sólo para evitar que se propaguen sus complejidades, sino para aprovechar la información específica que se pueda obtener en un momento determinado. El objetivo del uso de tal conocimiento es la *detección temprana de la ambigüedad*.

La detección temprana de ambigüedad significa para nosotros identificar un fenó-

meno textual en el cual una palabra o una expresión multipalabra señala de manera inambigua la estructura de predicados que esa unidad de texto contiene. Consideremos el siguiente ejemplo de especificación

$$\begin{aligned} &\textit{Si se inserta una moneda, el sistema devuelve un t\'e.} \\ &\textit{En cambio, si se proveen dos monedas retorna un caf\'e.} \end{aligned} \tag{4.2}$$

y asumamos completada la etapa de Preprocesamiento. El módulo de Extracción interactúa con la componente de Conocimiento Estático para proveerse de información que permita resolver fenómenos usando claves de decisión. Por ejemplo, diccionarios que almacenen categorías gramaticales y expresiones reconocibles en las estructuras sintácticas obtenidas por Preprocesamiento. Así, a partir de extraer la clave textual que identifica en el árbol T de tipo `Tree` la subordinación por condicional entre los predicados *se inserta una moneda* y *el sistema devuelve un té*, podemos luego implementar una heurística simple que interprete esa forma textual como tal relación de subordinación. La clave en cuestión podría ser el lema `si` y/o los no terminales `subord` y `ador` que se introducen durante el Preprocesamiento –y a los que accedemos en una instancia de `Payload` desde `category` y `subcategory`.

Sin embargo, la dificultad radica en un nivel más alto en la jerarquía ideal de dependencias para texto de entrada. La expresión *en cambio* está indicando una coordinación por disyunción entre los grupos de predicados A = *"si se inserta una moneda, el sistema retorna un té"* y B = *"si se proveen dos monedas se obtiene un café"*. Esa expresión, marca inconfundible de resolución de ese fenómeno dada la forma textual de esas oraciones, no es analizada originalmente por la herramienta del Preprocesamiento como tal conector de disyunción, sino como una frase preposicional dependiente del grupo verbal cuyo núcleo es *retorna*. La jerarquía ideal constituye una estructura compleja y no es la obtenida por nuestra herramienta.

No se trataría de un diccionario con expresiones el que provea de información para intentar resolver este caso, ya que no puede explorarse a T de forma tan simple: la presencia de nodos para la expresión *en cambio* en algún nodo descendiente del verbo núcleo de alguno de los grupos verbales presentes, y la correspondencia de esa expresión con una marca de disyunción en un diccionario, no es suficiente para deducir que esa expresión ocurría en el texto original justamente como tal marca entre A y B, y no en otro lugar de menor precedencia.

La estrategia que proponemos para resolver este fenómeno es reflejar de otra forma en el Conocimiento Estático la condición inequívoca de esa expresión como marca de inambigüedad. Por inspección de la herramienta seleccionada para abstraer como Preprocesamiento, sabemos que muchos de sus archivos internos de configuración son

transparentes a modificarse. Por lo tanto, puede extenderse un archivo de locuciones para indicar que durante el *dependency parsing* se use la expresión *en cambio* como marca de disyunción. Como consecuencia, la jerarquía de dependencias obtenida será la esperada y en T la etapa de Extracción distinguirá tal marca. Luego, la Interpretación será directa con una heurística simple para disyunción, similar a la que resolvía, párrafos atrás, la subordinación por condicional.

Una vez implementada esta extensión del archivo de locuciones, podemos pensar que, si bien ese fenómeno es ambiguo en el texto plano de entrada, técnicamente no es ambiguo en la Extracción ya que, en la jerarquía que ahora se construye, no hay dudas acerca de esa expresión: el Preprocesamiento la identifica como marca de disyunción y así es extraídas en T. O en todo caso, hemos detectado una ambigüedad muy tempranamente, a nivel de implementación del sistema y no durante la ejecución del mismo.

Ahora bien, consideremos la detección temprana de la ambigüedad tal como se pretende referir a un marco general de estrategias para enfrentar determinados fenómenos. Será poco probable que podamos identificar y resolver muchas más claves inequívocas de la manera que se explicó recientemente. Más aún, la enorme mayoría de los casos serán fenómenos de ambigüedad para los que no habrá claves inconfundibles en la forma textual. Deberá darse tratamiento a su resolución durante la ejecución de un análisis de una especificación de entrada, esto es, dinámicamente. Nuestra propuesta entra en acción.

El siguiente ejemplo apenas se diferencia del ejemplo 4.2, pero es un paradigma de uno de los conjuntos de fenómenos que se pretende estudiar:

Si se inserta una moneda, el sistema devuelve un té. (4.3)
Si se proveen dos monedas retorna un café.

¿Cómo se relacionan los dos grupos de predicados A = *"si se inserta una moneda, el sistema devuelve un té"* y B = *"si se proveen dos monedas retorna un café"*, correspondientes respectivamente a la primera y a la segunda oración? Pueden proponerse dos interpretaciones:

1 Secuenciación: Si se inserta una moneda, el sistema devuelve un té. **Y luego**, si se proveen dos monedas retorna un café.

2 Alternativa: Si se inserta una moneda, el sistema devuelve un té. **O bien**, si se proveen dos monedas retorna un café.

Si la máquina de café que se está especificando es capaz de retornar una u otra bebida por los respectivos precios mencionados, entonces ambas posibles interpretaciones

son válidas. La primera expresa una secuencia de dos implicaciones *"si se provee... entonces se retorna.."* independientes, y la segunda una alternativa entre las implicaciones. Una tercera posibilidad poco intuitiva sería igual que la primera, pero donde las implicaciones no sean independientes y fuera preciso que el sistema complete una entrega de té a partir de una moneda para luego realizar la entrega de café desde el ingreso de dos monedas. Es decir, algo de la forma

3 Condicional: Si se inserta una moneda, el sistema devuelve un té. Si se insertó una moneda y luego el sistema devolvió un té, entonces si se proveen dos monedas retorna un café.

La segunda interpretación es la que en general consideraríamos que el humano quiso expresar en la especificación, pero no podemos asegurar que resulte tan claro que la primera sea poco natural o improbable. En otros puede ser más evidente, pero no será evidente para el análisis automático hasta que no se dispongan de mecanismos que aseguren tal evidencia. El sistema debería estar en condiciones de detectar que posiblemente, en lugar de la interpretación básica como la de la primera propuesta, se trate de una disyunción y entonces intente resolver esta incertidumbre.

El módulo de Extracción debe reconocer la forma textual de dos oraciones consecutivas por punto seguido, cada una de las oraciones expresando una subordinación por condicional, como indicadora de esa situación. En este caso, no hay una expresión de marca ambigua, sino que se trata de una forma textual mucho más extensa. En lugar de punto seguido, podría presentarse, por ejemplo, una expresión como *pero*, o una expresión multipalabra. En cualquier caso, se trata de una forma textual de ambigüedad que podemos reconocer durante el estudio previo a la construcción del sistema –para así implementar estrategias que justamente la detecten–, pero no podemos resolverla inequívocamente.

La característica interactiva nos indica que debemos solicitar información al humano. Una situación como la del ejemplo 4.3 podría resolverse de la siguiente manera:

a) el módulo de Extracción detecta la presencia de esa forma textual ambigua;

b) se consulta en diccionarios del Conocimiento Estático las posibles interpretaciones que alguna vez se propusieron para esta forma ambigua, en este caso, conjunción o disyunción;

c) el módulo de Extracción encarga al módulo de Regeneración la recuperación de expresiones en lenguaje natural A_{regen} y B_{regen} para los grupos de predicados correspondientes a las expresiones originales A y B, respectivamente;

d) el módulo de Extracción encarga a la componente de Conocimiento Dinámico formular preguntas al humano, incorporando en ellas las claves que sí sean inconfundibles obtenidas de la consulta en el paso **(b)**. Se pregunta algo como: *En la siguiente expresión "A_{regen}. B_{regen}", ¿cuál de las siguientes quiso expresar?*

1 - A_{regen} ***Y luego*** B_{regen}

2 - A_{regen} ***O*** B_{regen}

e) según la respuesta obtenida, se obtiene una nueva especificación de requerimientos, y se vuelve con ella a la etapa de Preprocesamiento. Esta nueva especificación difiere de la original sólo en que ahora no presenta la clave ambigua sino una clave artificial inequívoca;

f) en una segunda iteración, el Preprocesamiento identifica, en la especificación modificada, la clave inconfundible para la construcción de la jerarquía de dependencias;

g) el módulo de Extracción detecta y obtiene esa clave en la estructura de árbol accediendo a inspeccionar los atributos de las instancias de `Payload`;

h) una heurística simple en el módulo de Interpretación puede resolver el caso de la conjunción cuando se satisface tal clave previamente extraídas, y análogamente otra resuelve la disyunción;

i) la ejecución sigue como en la secuencia lineal, con las posibles interpretaciones restantes, eventuales regeneraciones y finalmente la codificación del grafo.

Debe notarse que en el paso **(b)** estamos obteniendo información sólo orientativa, y no determinista o inequívoca, como se entiende será el caso abrumadoramente mayoritario de situaciones a resolver. Sin embargo, este conocimiento Estático puede modificarse como resultado de la respuesta obtenida en **(e)**, sujeto a decisiones que seleccionen sólo la información fiable desde el exterior como posible de extender el de nuestro sistema. De esto se trata la mencionada combinación de beneficios mutuos entre ambas componentes de conocimiento.

Como puede verse, la disposición de la respuesta provista por el humano en interacción desplaza el enfoque lineal de la secuencia de etapas hacia un tratamiento iterativo de los datos.

La estrategia de interacción que presentamos solicita respuestas al humano desde claves identificadas en la etapa de Extracción, como parte de la detección temprana de la ambigüedad. Análogamente, la información se podría solicitar a partir de mecanismos de decisión a nivel del módulo de Interpretación. En este caso, las ambigüedades son de mayor complejidad, ya que no se detectan a nivel de forma textual sino en la jerarquía

de dependencias sintácticas, por ejemplo, la ya explicada ambigüedad por adjunción de sintagma preposicional. A partir de un eventual conflicto entre la satisfacción de las condiciones de aplicabilidad de más de una regla de decisión, la interacción sería muy similar a la presentada. Si bien se corresponde a la demanda de un conocimiento diferente, la solicitud al humano debe ser siempre una que abstrae el fenómeno particular de ambigüedad que la dispara y sugerir posibles interpretaciones usando claves inconfundibles que luego se introduzcan durante la modificación de la especificación en el procesamiento iterativo.

4.7. Regeneración de Lenguaje Natural

(Volver al Índice general)

La etapa de Preprocesamiento, con el *dependency parsing* como tarea central a la misma, desordena el texto plano de la especificación de entrada para obtener las jerarquías de dependencias sintácticas. Tal como hemos presentado el diseño general del sistema, la misma etapa de Preprocesamiento se encarga de extraer automáticamente elementos de ese texto original y preservarlos como parte del Conocimiento Dinámico para un eventual proceso que hemos denominado Regeneración de Lenguaje Natural.

Una vez obtenidas los tokens de un predicado, se puede recuperar el orden lineal total entre dos palabras cualesquiera haciendo uso de estos elementos preservados: conjuntos de segmentos significativos del texto de entrada, cada conjunto definido por una longitud de ventana con la que determinar los segmentos. Se implementan un algoritmo de búsqueda sobre estos segmentos para establecer el orden entre un par de palabras.

Este módulo de Regeneración es invocado a la hora de obtener el grafo objetivo, ya que si bien se entrega una estructura codificada que abstrae los predicados textuales, un posible servicio posterior que nuestro sistema brinde sería justamente el de decodificación de las etiquetas de arcos. Una aplicación externa puede transformar el grafo codificado en otro grafo que preserva un subconjunto de las etiquetas codificadas. Luego, tras la decodificación, un grafo decorado con los predicados en lenguaje natural brinda al humano toda la expresividad apropiada para su interpretación.

Asimismo, la Regeneración es fuertemente utilizada para reconstruir el orden lineal original no solo de predicados sino de secuencias de oraciones en las que se identifican formas textuales ambiguas. Con estas secuencias regeneradas se logra solicitarle efectivamente al humano la interacción que se requiera. Y también se obtiene eventualmente una nueva especificación, levemente distinta a la original, que introduce información desde la respuesta humana en interacción y con la cual se retroalimenta iterativamente

desde el Preprocesamiento.

Capítulo 5

Instanciación y Análisis de Fenómenos

(Volver al Índice general)

Se presentan en este capítulo los detalles y decisiones específicas que conforman la instanciación de nuestro sistema.

Previamente en el despliegue de nuestra metodología, hemos identificado el problema de la especificación de requerimientos de software en lenguaje natural para obtener una representación general de su información. Definimos objetivos del trabajo, y analizamos otros trabajos relevantes para identificar sus desventajas y seleccionar las mejores características.

Tras la exploración bibliográfica, identificamos nuevas restricciones impuestas sobre los recursos de ejemplos de los que disponíamos, y se decidió adoptar un enfoque interactivo para dar cuerpo a las estrategias de interpretación de especificaciones. Luego, diseñamos y desarrollamos una arquitectura del sistema, tanto en un alto nivel como en uno de mayor detalle. Esta arquitectura es en definitiva un esqueleto o macroestructura del sistema, quedando por determinar varias partes que la definen.

Como veremos a continuación, se han seleccionado y estudiado varios fenómenos semánticos, lingüísticos y de interacción para establecer el sentido práctico de la arquitectura y así dar tratamiento a especificaciones concretas. Denominamos a estas decisiones como la *instanciación* del sistema. Es una etapa previa a la implementación, ya que ésta última refiere a un nivel inferior de abstracción, el de desarrollo de código, mientras que nuestra instanciación está particularizando mucho de las lógicas y procesos que los módulos involucrarán. Será claro, luego de entender los fenómenos propuestos, que para la misma arquitectura del capítulo anterior se podía considerar un conjunto de fenómenos distinto al actual, y el sistema podría ser implementado en consecuencia resolviendo esos fenómenos deseados. Esto es, otra vez, lo que se considera

la instanciación.

En suma, se distinguieron los siguientes criterios para seleccionar los fenómenos a estudiar:

- frecuencia del fenómeno;
- utilidad para nuestro objetivo;
- factibilidad del tratamiento automático.

5.1. Fenómenos semánticos

(Volver al Índice general)

Como ya se ha explicado anteriormente, el objetivo central del sistema presentado en este trabajo es obtener, a partir de una especificación de requerimientos de software expresada en lenguaje natural, un grafo de representación de la secuencia de acciones que se describe en ese texto de entrada. La reiteración de este objetivo viene al caso de hacer claro que pretendemos identificar cada predicado e interpretar sus relaciones en tal secuencia: no es nuestro propósito obtener representaciones de la semántica de esos predicados. Es decir, no se busca, por el momento, determinar sus significados con algún grado de precisión, en particular, por ejemplo, mediante reconocer de alguna forma que dos acciones textualmente distintas significan lo mismo y unificar las representaciones asociadas de tales predicados sinónimos.

De todas formas, sí decidimos emprender el desafío de intentar la captura de algunos fenómenos como los que se describen a continuación. Reciben la denominación de fenómenos semánticos por tratarse de características relativas al significado de lo que se expresa en la especificación, como parte de un propósito de enriquecer la representación en grafo a obtener. Las unidades de información consideradas son diversas según el fenómeno en cuestión: se han analizado desde elementos componentes de un predicado hasta estructuras en la secuencia de acciones.

5.1.1. Cursos alternativos en el caso de uso

Una decisión de diseño muy importante es la de preservar un formato para escribir el caso de uso de entrada tan cercano al nivel del humano como sea posible. Aspecto distinguido de nuestro sistema, provee acceso a aquellos humanos no educados en conceptos de elicitación de requerimientos.

Aún así es posible realizar un manejo más inteligente de la entrada de texto plano. Un formato simple pero a la vez poderoso que se propone es el siguiente:

- para el flujo normal de acciones, un paso de la secuencia por cada línea, separada de la siguiente por un marcador de nueva línea;
- un número de paso al principio de cada línea;
- una línea especial formada por alguna secuencia de caracteres reservados que no aparezcan en las posibles palabras de la especificación. Esta línea especial separa arriba el flujo normal de acciones y abajo los cursos alternativos. También separa cada secuencia de curso alternativo de la siguiente, si es que hay más de un curso de alternativas;
- para cada curso alternativo, un formato similar al del flujo normal.

El siguiente es un ejemplo de un caso de uso basado en este formato de entrada:

1 La máquina imprime un cartel solicitando elegir una bebida.
2 El usuario inserta una moneda.
3 La máquina retorna un té.
*****
2 El usuario inserta una moneda.
3 La máquina retorna un café.
*****
. . .

Aquí asumimos, sólo a modo de ejemplificación del formato propuesto, que el curso normal es obtener un té. En general, los cursos alternativos expresan situaciones eventuales pero menos frecuentes que el flujo normal o principal, a la vez que reducen apenas el costo que se tendría si cada curso alternativo requiriera constituir un nuevo caso de uso con las acciones ocurriendo repetidas en varios de ellos.

Dada una representación en texto plano como la anterior, debemos incorporar algún tipo de interpretación de los elementos de formato durante la etapa de Preprocesamiento. Un formato ligero como éste resulta beneficioso tanto para el humano autor del caso de uso como para nuestras estrategias de interpretación.

Para el humano, la distinción entre curso normal y cada uno de los alternativos resulta clara y directa, y cada secuencia es también simple como enumeración de pasos. Además se adecua a algunos estándares y convenciones que establecen una estructura en cursos como ésta.

Desde el punto de vista del sistema, pueden capturarse las alternativas –a ser representadas como ramificaciones en el grafo– de manera mucho más eficiente y con mucho

menos error que si se tuviera que dar tratamiento a las ambigüedades propias de sentencias más complejas como las condicionales de la forma *"si... entonces..."* que contiene el mismo ejemplo pero en el formato textual de 4.3:

Si se inserta una moneda, el sistema devuelve un té.

Si se proveen dos monedas retorna un café.

5.1.2. Tipo de predicado: input y output

Cada predicado a identificar forma parte de un conjunto de acciones que dan cuenta de las posibles interacciones, en un escenario particular, entre un agente, generalmente humano, y el sistema que se especifica. Más allá de las estructuras lingüísticas que lo contienen, puede pensarse, a partir de estos términos de interacción, que un predicado se clasifica como de tipo input o de tipo output.

Un predicado input es uno que se corresponde con una acción simple llevada a cabo por el agente humano que justamente provee un elemento de estímulo al sistema. Y un predicado output es uno que representa una respuesta del sistema al estímulo del humano. Así, en el paradigmático ejemplo 4.3, el predicado *"insertar una moneda"* es de input y *"devolver un té"* es de output.

Como se sugirió recién, un predicado de output se corresponde con una respuesta del sistema *al estímulo* del humano. Aunque en la enorme mayoría de los casos se entienda que eso significa lo mismo que "una respuesta del sistema *al humano*", no es necesariamente así.

Esta caracterización de predicados puede enriquecer la representación que finalmente se obtiene en forma de grafo. Idealmente, los tipos deberían ser determinados inequívocamente o con una alta precisión. Para identificar los tipos correctos de predicados, la estrategia baseline que se implementó –y que al momento de escribir este trabajo es la usada por el sistema– es clasificar por el verbo conjugado que domina al predicado.

Inicialmente, para cada uno de los dos tipos, se crea –como parte del conocimiento Estático– un lexicon de verbos que se consideran inconfundibles respecto a esta clasificación. Luego, para cada verbo se agregan sinónimos obtenidos manualmente por inspección en diccionarios. Por ejemplo, el verbo *retornar* puede ser entendido como inequívoco de un predicado de output, mientras que *introducir* lo sería para input.

Además, se utiliza el sustantivo núcleo del sujeto sintáctico de ese predicado cuando tal sustantivo sea también marca inconfundible del agente que realiza la acción. Por ejemplo, *"usuario"* indica que es un predicado input y *"sistema"* indica output. Se construyen sendos lexicones de sustantivos para almacenar estas claves.

En general, estas dos heurísticas simples de verbos y sustantivos núcleos parece resolver los tipos de una buena cantidad de los predicados involucrados. El humano que escribe un caso de uso suele usar muy pocos verbos para explicar un gran conjunto de acciones que se disponen alternados en una secuencia de interacción entre un par de agentes. Entonces, una vez que se asocia un tipo a un verbo en particular, es posible resolver varios predicados que lo contienen. Sin embargo, para algunos verbos no es directa una clasificación así.

Por último, la estrategia baseline se complementa agregando al lexicon apropiado de verbos el verbo conjugado que acompaña en el predicado a tal sustantivo cuando éste se agrega a su lexicón de sustantivos.

La automatización de la búsqueda de sinónimos es una tarea compleja dado que al tratarse de recursos digitales, por ejemplo documentos web, los diccionarios presentan diferentes formatos entre sí y por lo tanto la inspección manual termina siendo más eficiente que la codificación de un script inspector. Luego propiamente como fenómeno lingüístico, explicaremos otras limitaciones en la automatización de esta estrategia.

5.1.3. Alcance de la validez de un predicado

Un interesante fenómeno a analizar es el de alcance de la validez de un predicado: determinar sobre qué conjunto de estados se satisface.

Tal descripción del fenómeno parece sugerir la necesidad de definir formalmente a un estado y a la satisfacción similares a los conceptos de satisfacción lógica de un predicado; sin embargo, relajamos la consideración de tales definiciones. Dada una estructura sintáctica compleja para un grupo de predicados, es factible de ser representada en un grafo objetivo como un par de estados nodos y un arco entre ellos decorado con tal grupo. Luego la fragmentación de la estructura en predicados dispara la creación de estados intermedios, y los arcos reflejando apropiadamente la relación entre los predicados identificados.

El horizonte ideal del tratamiento de este fenómeno es que cada estado tenga asociado un conjunto de predicados que se satisfagan en él. Y añadiendo en el ideal posible que los predicados generen semánticas para luego unificarlas si son similares bajo algún criterio. Por ejemplo, unificar como la misma acción la *"encender la máquina"* y *"prender la máquina"*. Análogamente para *"el usuario inserta una moneda"* y *"la moneda es insertada por el usuario"*.

El estadío actual de la instanciación para este fenómeno es un baseline que consiste en asumir como alcance de un predicado simplemente al estado al que se llega una vez terminado ese predicado. Este baseline brinda la máxima precisión aunque posiblemente sacrificando un poco de cobertura.

Claramente, parte fundamental de las estrategias para mejorar el tratamiento de nuestro sistema sobre este fenómeno están relacionadas con determinar semántica de los predicados involucrados, que no es precisamente nuestro propósito en este trabajo. Con ese enfoque, sabiendo los significados similares de *"no emitir"* y *"dejar de emitir"* sería directo que el alcance de *"El sistema emite un sonido"* no puede ir más allá del estado posterior a la finalización de *"El sistema deja de emitir el sonido"* en un caso de uso como éste:

1 El sistema emite un sonido y muestra un cartel de alarma.
2 El usuario presiona el botón etiquetado como "Desactivar".
3 El sistema deja de emitir el sonido y muestra un cartel de OK.

Además de esto, se agrega el posible estudio del concepto de acción finalizada. ¿Cuándo se puede afirmar que una acción ha concluido? ¿Qué subacciones componentes permanecen implícitamente activas? El marco lingüístico correspondiente, y por tanto su tratamiento y aplicación, exceden los objetivos de este trabajo.

5.1.4. Pre-condiciones y post-condiciones

Un campo de información muy frecuente en las convenciones de especificaciones es el de pre-condición y post-condición que deben satisfacerse para el caso de uso descrito. En nuestro trabajo, descartamos por el momento elicitar ese tipo de información.

Si bien son relativamente fáciles de incluir en futuros desarrollos, el principal motivo para no exigir estas condiciones es que el usuario no educado no maneja estos conceptos en su especificación funcional. Y relacionado con esto, la ausencia de tales condiciones se condice fuertemente con nuestro formato flexible para ingresar el caso de uso. Eran justamente esas condiciones algunas de las partes más características de los formatos rígidos que se mencionaron al inicio de este trabajo.

La incorporación de pre y post-condición en la confección de la especificación en lenguaje natural puede tener impacto en el tratamiento ideal del fenómeno de los alcances de predicados de la subsección previa.

5.2. Fenómenos lingüísticos

(Volver al Índice general)

Nos proponemos estudiar algunos fenómenos lingüísticos, entre ellos, y en una primera subsección muy importante, los relativos a la extracción e interpretación de

expresiones ambiguas mediante las estrategias de una arquitectura de sistema *Human-in-the-Loop* antes mencionadas.

5.2.1. Interacción y Heurísticas Ad-hoc

En el capítulo 2 se mencionaron algunos trabajos relevantes con apartados para estudiar alguna clasificación de ambigüedades. Sería útil pensar en agrupar con algún criterio las diversas ambigüedades que se pueden presentar. En el trabajo de (Chantree, Nuseibeh, de Roeck, and Willis 2006), las ambigüedades presentes en requerimientos de software expresados en lenguaje natural se clasifican según la magnitud de una eventual interpretación errónea para cada una de ellas. También se proponen técnicas para detectar las ambigüedades nocivas de entre las ambigüedades triviales o de interpretación directa.

El problema de las ambigüedades es central a nuestro sistema. Luego, resultaría muy conveniente darle tratamiento a partir de un enfoque sistemático que contemple una gran cantidad y variedad de casos a considerar. Sin embargo, como ya se adelantaba en el capítulo 3, se dispone de un reducido volumen de corpus a analizar. Esta restricción, considerada en relación al hecho de que nuestro enfoque se basa fuertemente en conocimiento asociado a expresiones lingüísticas, impone luego la necesidad de una estrategia general de heurísticas ad-hoc.

En nuestro sistema, una heurística es esencialmente una regla de decisión tal que si un elemento satisface su condición de aplicabilidad, entonces la misma es aplicada sobre una estructura para la que ese elemento es significativo. Un elemento sujeto a validar aplicabilidad de una regla se constituye por una o más claves textuales identificadas en la etapa de Extracción, durante la cual se hace uso eventualmente de mecanismos de interacción para obtener información desde el agente humano. Una estructura interpretada por la heurística puede ser un predicado, o un grupo de predicados en los que se resuelve la relación entre los mismos.

El tipo de estructuras de relación de predicados es sin duda el de mayor abundancia y complejidad. Dada su naturaleza lingüística de relación interclausal, se alejan del nivel sintáctico hacia mayores abstracciones de significación, lo que implica una enorme dispersión, inevitable entre los escasos casos observados, y en particular, muy rica –y por lo tanto problemática– en ambigüedades. Como un ejemplo de estas complejidades fundamentales que intentamos resolver, pensemos en varias de las formas con las que se puede expresar la relación condicional o de implicación entre el predicado antecedente *"El usuario inserta dos monedas"* y el consecuente *"La máquina retorna un café"*:

- 1 *El usuario inserta dos monedas.*

2 *La máquina retorna un café.*

- *El usuario inserta dos monedas. La máquina retorna un café.*
- *Si el usuario inserta dos monedas, la máquina retorna un café.*
- 1 *El usuario inserta dos monedas.*

  2 *Si el usuario insertó dos monedas, la máquina retorna un café.*
- *La máquina retorna un café si el usuario inserta dos monedas.*
- *Cuando el usuario inserta dos monedas, la máquina retorna un café.*
- *La máquina retorna un café siempre que el usuario inserta dos monedas.*
- *El usuario inserta dos monedas y la máquina retorna un café.*
- *El usuario inserta dos monedas. Luego, la máquina retorna un café.*

Aunque algunas de estas formas serían probablemente reducidas o divididas durante la confección de una especificación definitiva, no debe perderse de vista la ambigüedad que cada forma reviste, así como la eventual multiplicidad de expresiones para una misma idea simple. Multiplicidad que en un estudio como el que llevamos a cabo, más allá del pequeño corpus disponible, significa una gran dispersión de datos característica del lenguaje natural.

Inicialmente, el sistema está provisto de heurísticas construidas sólo en base a la información que se inspecciona en los ejemplos de análisis. En este sentido es que cada heurística es pensada como ad-hoc, ya que resuelve un patrón textual muy específico. Luego, a partir de encontrar regularidades en los elementos que activan su aplicación, algunas heurísticas pueden generalizar tal elemento o unificarse con otras en ese patrón de aplicabilidad. Claramente, el incremento en el volumen de los casos observados tendrá impacto significativo tanto en la precisión con la que se capturan patrones como en la posibilidad de generalizar los fenómenos.

En resumen, se parte de las siguientes hipótesis para dirigir la estrategia de diseño de heurísticas ad-hoc:

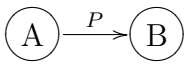
- corpus disponible de pequeño volumen;
- ambigüedades presentes en la gran mayoría de expresiones, como característica del lenguaje natural;
- necesidad de resolver inicialmente cada patrón en particular con una solución muy especializada;

- alta probabilidad de generalizar algunas heurísticas identificando regularidades en los elementos de activación de las reglas, pero no lo suficiente para que éstas dejen de ser parcialmente especializadas;
- las especificaciones mediante casos de uso siguen algunas convenciones, y por lo tanto las expresiones a tratar conforman una secuencia de acciones generalmente simples para identificar cada predicado;
- necesidad de suponer complejidad en las expresiones suficiente como para dar una heurística general baseline que dé robustez a nuestro sistema en la interpretación de las relaciones entre predicados;
- horizonte de mayor sistematización en el tratamiento de los fenómenos a partir de una mejor clasificación de las ambigüedades durante la etapa de Extracción, como consecuencia de una eventual disponibilidad a futuro de un corpus mucho más voluminoso.

Pasemos ahora a considerar el conjunto de heurísticas propuestas. Se trata de un conjunto de reglas establecidas manualmente que se basan en la presencia de ciertas claves textuales superficiales –palabras o expresiones clave, estructuras sintácticas– como patrones de activación para la aplicación de una regla en particular. Estas claves textuales proveen la mayor parte del conocimiento heurístico.

Supongamos que S_1, S_2 son segmentos de texto, y P_i es la representación para los predicados obtenidos desde S_i, $i = 1, 2, ...$ Generalmente pensamos las representaciones P_i como de estructuras de predicados o acciones, ya que pueden tratarse no sólo de predicados simples, sino que el correspondiente segmento S_i corresponde a un grupo de predicados con cierta relación a interpretar recursivamente.

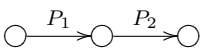
En el grafo, mediante un arco dirigido desde A a B decorado con P se representa la situación de un estado A previo a la realización del predicado P y un estado B posterior al mismo. Gráficamente:



De acuerdo a los fenómenos identificados, hemos propuesto las siguientes heurísticas para resolver la relación entre predicados.

1) Baseline: De Precedencia. Se trata de una heurística clásica debida a (Grosz and Sidner 1986), que utiliza la precedencia lineal de los segmentos de texto. Si un segmento de texto S_1 precede a otro S_2, eso implica que los predicados que hemos obtenido de S_1 tienen que estar satisfechos para que se pueda tratar

de satisfacer los predicados que hemos obtenido de S_2. Luego, el patrón textual S_1 S_2 de concatenación de tales segmentos implica una relación de la forma $P_1 \rightarrow P_2$. Para ello, una vez identificado cada predicado, la relación entre sí queda determinada por la precedencia dada sobre la estructura de datos T de tipo TREE construida oportunamente a partir del *dependency parse tree*.

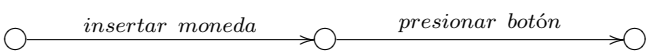


Ésta es la relación por defecto entre predicados, es decir, establecemos la precedencia como heurística por defecto para garantizar la robustez del sistema. Por encima de ésta se construye una jerarquía de reglas más especializadas que claramente se intentan aplicar antes que el baseline.

2) De Conjunción. Una relación de conjunción entre P_1 y P_2 se interpreta con esta heurística identificando justamente una clave de conjunción sobre T. La misma puede tratarse de una palabra o expresión que determine inequívocamente una conjunción, o bien de una palabra artificial previamente conocida que hemos introducido en la etapa de Extracción –tal como se explicara en el capítulo anterior.

Dado que se deben satisfacer ambos predicados en conjunción, la representación se corresponde con una concatenación de los mismos, en algún orden a convenir. Si se conviene el orden lineal, se tendría una representación similar a la obtenida por regla de precedencia. Este orden es conveniente ya que en general la conjunción en lenguaje natural trae implícito un orden de acciones, es decir, se está también pensando implícitamente en que el primer miembro de la conjunción se debe satisfacer para luego realizar el segundo.

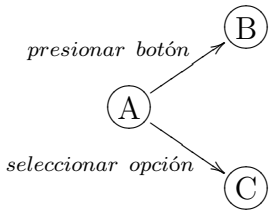
Por ejemplo, una estructura de datos que se corresponde con la forma textual *"El usuario inserta una moneda y presiona el botón rojo"* se interpreta inequívocamente como una conjunción vía la clave *y*. Podría, tal como se explicaba recién acerca del orden implícito, en interactuar con el usuario sugiriendo una posible implicación, pero al final la representación a obtener será equivalente. El subgrafo sin codificar para ese ejemplo es:



3) De Disyunción. Análoga en el tratamiento de las claves a la recién desarrollada regla de conjunción. Tal como se sugiere en (Bajwa, Lee, and Bordbar 2012), en un texto en lenguaje natural una disyunción puede ser inclusiva o exclusiva

desde el punto de vista de la lógica de predicados. A su vez, puede tratarse de una disyunción entre sustantivos o adjetivos, o bien entre dos frases verbales. Nuestra heurística debe resolver sólo el segundo caso, el caso de interés, y por los ejemplos observados, asumimos que en el contexto del lenguaje natural un humano expresa una disyunción entre frases verbales como disyunción exclusiva: *"El usuario presiona el botón rojo o selecciona la opción <Volver>"* .

Identificados los predicados en disyunción P_1 y P_2, la representación es justamente de ramificación desde un estado A a uno B decorado por P_1, y desde el mismo A a uno C decorado por P_2.



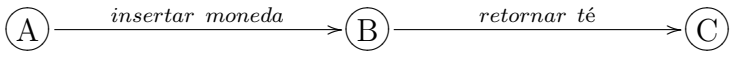
En el caso de las últimas dos heurísticas explicadas, debe tenerse en cuenta que estamos resolviendo inequívocamente sólo mediante la extracción de claves inequívocas, pero que la identificación de conjunción y de disyunción en estructuras más complejas y con presencia de claves ambiguas es un problema muy importante en el tratamiento del lenguaje natural. Luego, un gran volumen de las expresiones identificadas como inequívocas en realidad son reemplazos realizados durante la etapa de Extracción a partir de información en interacción, sobre expresiones ambiguas en el texto de entrada.

4) De Implicación. Esta regla captura algunos patrones de implicación utilizando como elemento de activación de aplicabilidad una combinación de palabras o expresiones claves que indican condicional, por un lado, y por el otro las etiquetas de no terminales que provee el parseo de dependencias sintácticas durante el Preprocesamiento y que han sido almacenadas en los atributos ya explicados de la clase `Payload`.

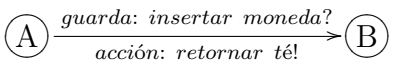
Si se tiene, como situación más frecuente, el patrón de implicación

$$Si\ S_1,\ S_2$$

–como en el ejemplo paradigmático *"Si el usuario inserta una moneda, la máquina retorna un té"*–, la representación obtenida es de la forma de un arco dirigido desde el estado A al estado B decorado con P_1 –la estructura de predicados correspondiente al segmento S_1, esto es, la acción antecedente– y luego desde B a un estado C vía etiqueta P_2 –la consecuente. Para ese ejemplo, gráficamente:



Una representación alternativa podría ser con sólo dos estados A y B, respectivamente, previo y posterior a la satisfacción o realización del predicado $P_1 \Rightarrow P_2$, y un arco entre ellos decorado con una etiqueta compuesta de un campo de guarda o condición, P_1, y otro de acción propiamente dicha, P_2. Por ejemplo:



Como se ve, esta decisión involucra disponer de etiquetas que distingan acciones más complejas precedidas por guardas. La representación que decidimos adoptar es más simple ya que sólo considera acciones, incluyendo a la de guarda como una forma de acción según la idea de predicado que se introdujo oportunamente; se ve satisfecha o realizada en el estado intermedio, y desde el cual es posible alcanzar el tercero a través de la consecuente. Denominamos *inflado de implicación* a esta estrategia de representación.

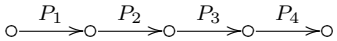
Al momento de redactar este trabajo, éstas son las reglas que se encuentran implementadas en el sistema. La versión disponible de cada una es la obtenida luego de algunos pasos de generalización de los patrones que las activan. En particular, la regla de implicación, a partir de unificar varias reglas similares muy específicas en sus patrones.

En relación a esto, y teniendo en cuenta el ejemplo que mostraba la multiplicidad de formas con las que se puede expresar una relación de implicación simple, hemos identificado además otros fenómenos más complejos. Si bien se asumen poco frecuentes dada la hipótesis de uso de convenciones en la redacción de un caso de uso, no deben descartarse en pos de proveernos de mayor robustez, sobre todo si se tratan de especificaciones no definitivas en donde pueden presentarse con mayor frecuencia. Para ellos, se proponen las siguientes interpretaciones:

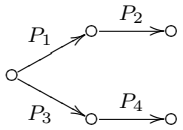
5) Dos condicionales consecutivos. El patrón textual de la forma

$$Si\ S_1,\ S_2.\ Si\ S_3,\ S_4$$

podría tener, según la heurística baseline, una interpretación *naïve* como ésta:



pero una interpretación correcta que proponemos consiste en asumir que una clave como la del punto y seguido indica alternativa entre el par de condicionales:



Más bien, la clave es la ausencia de una expresión conectora. Análogamente se tratan otros nexos asumidos como inequívicos. Una solución más laxa sería llevar la extracción de esta clave a la modalidad de interacción. Esta estrategia es aplicable a muchas otras situaciones textuales; puede reducir en algunos casos el error aunque aumenta el flujo de información y tiempo requeridos para la interacción.

En presencia de otros nexos, la interacción se vuelve más necesaria. Por ejemplo, si el par de implicaciones está ligado por una forma como ésta:

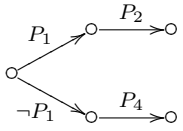
$$Si\ S_1,\ S_2,\ pero\ si\ S_3,\ S_4$$

si bien el *pero* es un nexo coordinante de conjunción, la intención del humano con esta estructura de acciones es claramente la de expresar una disyunción.

Si en particular, el segmento S_3 en el primer patrón de condicionales consecutivos es igual a *no*:

$$Si\ S_1,\ S_2.\ Si\ no,\ S_4$$

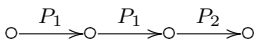
entonces interpretamos el predicado correspondiente como la negación del consecuente del primer condicional:



6) Solapamiento. Un patrón textual de la forma

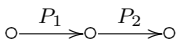
$$S_1.\ Si\ S_1,\ S_2.$$

puede interpretarse inicialmente por regla de precedencia como la siguiente representación.



Se trata de un solapamiento que intenta resaltar que una estructura de acción P debe ser realizada, y una vez concluida o satisfecha puede pasarse entonces a realizarse la siguiente. Así entendida, es la situación de toda acción a especificar aquí, en el sentido de que en la secuencia de acciones intercaladas entre los agentes esperamos naturalmente concluir una acción corriente para continuar con el resto. Puede ser parte de los recursos expresivos del humano que cae en estas redundancias ya que no utiliza un enfoque lógico en su disposición de acciones. La

interpretación propuesta es la que suprime el predicado redundante, de manera que se vuelve un caso de heurística de implicación:



5.2.2. Enriquecimiento de Lexicon para Tipos de Predicados

Más allá de la complejidad de automatización por formatos heterogéneos mencionada en la subsección 5.1.2, las dificultades más importantes para la estrategia baseline de construcción de lexicones de verbos son limitaciones en la automatización de tal enriquecimiento propias de un tratamiento de la sinonimia tan simple como el propuesto. En este sentido, se observa que:

1) En la red de sinónimos, el camino del verbo A al verbo B es en general uno tal que si bien cada par de nodos adyacentes son sinónimos, se pierde entre esos extremos A y B la sinonimia que nos interesa como clave de clasificación de predicados, aún cuando hay pocos nodos intermedios. Por ejemplo, en el camino

$A = proveer \rightarrow retornar \rightarrow devolver \rightarrow regurgitar = B$

2) Un mismo verbo puede ser a la vez posible indicador de clasificación de ambos tipos, por ejemplo *"solicitar"* indica un input de un usuario que consulta al sistema o un output de un sistema que requiere una respuesta al humano; incluso un output de un sistema que consulta a otro sistema externo.

En futuras secciones propondremos algunas mejoras a esta estrategia.

5.2.3. Portabilidad a Otros Idiomas

Una de las características que se propusieron como distintivas de nuestro sistema es la de intentar construir esta herramienta de interpretación de especificaciones para el idioma español. Sin embargo, también se decidió estudiar la posibilidad de analizar estos fenómenos antes desarrollados partiendo de casos de uso en otros idiomas. Nuestro interés está en detectar los aspectos del sistema más dependientes del lenguaje y a partir de eso, pensar en las adaptaciones apropiadas para su extensión a otros idiomas.

El inglés fue el primero de los lenguajes en elegirse para instanciar. Se reestructuró parte de la implementación del sistema, parametrizando por idioma inicialmente la etapa de Preprocesamiento. Se preservó la herramienta FreeLing 3.0 que provee los servicios necesarios para varios idiomas, entre ellos el inglés. Se observó que los árboles de parseo de dependencias obtenidos por la herramienta en los lenguajes español e inglés:

- presentan diferentes conjuntos de etiquetas para muchas de las categorías, en particular varias de las más comunes. Por ejemplo, la etiqueta `grup-verb` que en español demarca una frase verbal, en inglés es `clause`.

- no son isomorfos. Aún en una frase de estructura sintáctica y vocabulario simples, suficiente para dar la frase correspondiente al otro idioma como traducción palabra a palabra de la primera, las jerarquías de dependencias son diferentes.

Estas observaciones justifican las decisiones de diseño propuestas para una extensión a otro idioma. Más adelante hablaremos de las modificaciones posibles a nivel de Preprocesamiento, pero ahora se hace hincapié en

Se propone una parametrización por idioma sobre algunas de las estructuras de conocimiento que se presentan luego en la sección 5.3, como por ejemplo los diccionarios de lexicones para tipos de predicados. Las estructuras en cuestión tienen preponderancia en los módulos de Extracción y de Regeneración. Esta parametrización significa simplemente que para cada idioma existe una estructura análoga de conocimiento, cuyo volumen de contenido varía según el lenguaje al que se corresponde.

Respecto a las heurísticas ad-hoc del módulo de Interpretación, dado el no isomorfismo de los árboles, deben hacerse estrategias específicas para cada idioma. Ahora bien, existen clases abstractas que agrupan las estrategias que dan tratamiento a los fenómenos con la misma semántica. Por ejemplo, la heurística para identificar implicaciones, más allá de las etiquetas y expresiones involucradas y otras particularidades, es la misma para los diferentes idiomas: deben identificarse un predicado antecedente y uno consecuente, y resolver la representación de su relación. Además, todo el esquema lógico de la estrategia y el manejo de estructuras de datos internas serán análogos entre idiomas. Es por esta razón que para las heurísticas de Interpretación no se hace una parametrización que las agrupe por idioma. Las heurísticas se disponen según la expresión o patrón lingüístico que tratan, y dada una heurística en particular, algunas de sus decisiones internas se ven determinadas por condicionales de idioma. Esta aproximación es más razonable dado esta jerarquía o relación por semántica, mucho más fuerte y sistemática que las especificaciones menores por lenguaje. Este enfoque además se estructura mejor en código, más directo, más factorizado y corto.

5.3. Fenómenos y Componentes de Conocimiento

(Volver al Índice general)

Hemos presentado los fenómenos semánticos y lingüísticos que se seleccionaron para formar parte de la instanciación de la arquitectura general. También, hemos desarrol-

lado una implementación concreta de este sistema, y para completar ésta, necesitamos exponer la representación de las componentes de Conocimiento. Se han introducido varias sugerencias a lo largo del trabajo, pero es apropiado realizar una enumeración de las estructuras de datos que se proponen para instanciar tal conocimiento en esta implementación.

Además, revisamos algunos detalles de la relación entre los fenómenos a estudiar y las estrategias interactivas de obtención de información.

5.3.1. Estructuras *Estáticas*

La componente de conocimiento Estático se ha diseñado pensando en varios niveles de contexto del conocimiento. Distinguimos los niveles de Sistema, de Proyecto y de Caso. Este diseño por niveles expresa que, inicialmente, un subconjunto del conocimiento es un marco de información para el funcionamiento del Sistema en general, por ejemplo, los lemas o las categorías de dependencias sintácticas con las que se dispara la aplicación de una heurística en particular.

Por otro lado, datos como los lexicones de verbos y de sustantivos para determinar los tipos de predicados pueden tener una parte incluida en el nivel de Sistema, pero otra parte es más dependiente del Proyecto en particular que se está especificando. Como un ejemplo de esto, podemos pensar en el nombre propio del software apareciendo en lugar de *"el sistema"* como núcleo del sujeto que forma parte de un predicado realizado justamente por el sistema, como en *"La SuperDrink Machine 3000 retorna un té"*. Otra situación es la de una especificación de software donde el sistema es denominado "aplicación", "máquina", y/o el usuario en realidad es un "cliente", "vendedor", por mencionar algunos. Es decir, en cada proyecto en especificación existe un vocabulario para ciertos elementos importantes que aparecen repetidamente en varios de sus casos de uso.

Por último, pensamos en la posibilidad de un conocimiento muy específico, que dependa del caso de uso, en estrecha relación con la información interactiva que el humano pueda proveer.

Claramente, a medida que este conocimiento se considera más y más específico, se vuelve menos estable y persistente, y por lo tanto más sujeto a modificaciones temporales. O pensado de otra manera, es justamente menos generalizable para todo el sistema, no es duradero como conocimiento sólido que puede contribuir a futuro sobre otros sistemas de software a analizar, y sólo tendrá una vida útil durante la interpretación de esa instancia de especificación en particular.

Desarrollados ya los fenómenos a considerar, y explicado el diseño por niveles, pasemos a describir las estructuras de datos con las que se representa el conocimiento

Estático en la implementación.

A nivel de conocimiento de Sistema, la primera estructura de conocimiento es el mencionado archivo de locuciones de la herramienta abstraida por el módulo de Preprocesamiento. Como se comentaba anteriormente, las expresiones que allí se dispongan se asumen de larga persistencia, prácticamente permanentes en el sistema, y se agregan durante la implementación del sistema. A su vez, se complementa con diccionarios de expresiones para permitir eventualmente una mayor flexibilidad a la hora de extender ese conocimiento durante la ejecución del mismo.

Además, a este mismo nivel de Sistema se han construido los siguientes diccionarios.

- `d_io_lexicon`, diccionario de palabras determinantes de tipos de predicados. Se indexa por claves de tipos de predicados, input o output, y donde cada valor es un diccionario de lexicones para ese tipo, en consonancia con lo desarrollado en la subsección 5.1.2. Cada diccionario de lexicones se indexa por claves de categorías, hasta ahora sólo verbo o sustantivo, y el correspondiente valor es una lista de palabras de esa categoría que han sido seleccionadas como determinantes de ese tipo de predicados. Existe una instancia de estos diccionarios por cada idioma a considerar.

- `d_nonterm_tags`, de etiquetas no terminales. Se corresponde a las etiquetas con las que se construye el árbol de parseo de dependencias. Estas etiquetas contienen información respecto a las categorías morfosintácticas y la función semántica de cada palabra, y, como ya se explicara, son utilizadas por las heurísticas para reconocer los patrones y expresiones lingüísticas que activen su aplicación. Existe una instancia de este diccionario por cada idioma a considerar. Para cada idioma, su diccionario tiene el mismo conjunto de claves de categorías a considerar; los respectivos valores son las etiquetas concretas que la herramienta dispone en el árbol según el idioma del texto de entrada.

- un diccionario *wrapper* que se indexa por clave de idioma y le asocia como valor un nuevo diccionario `d_stat_<idioma>`. Éste último tiene el mismo set de claves para todo idioma, y cada diccionario en particular guarda como conjunto de valores a los diccionarios antes mencionados. Así, el diccionario *wrapper* implementado tiene la siguiente jerarquía –las claves siempre son cadenas, en particular, las claves de idiomas como “es” para español o “en” para inglés.

```
{ "es" : { "nonterm_tags" : { "grup-verb" : "grup-verb",
                              "sn" : "sn",
                              ...
                            },
```

```
            "lexicons" : { "input" : { "nouns" : ["usuario", "cliente"...],
                                       "verbs" : ["presionar", "insertar",...]
                                     },
                           "output" : ...
                         },
          },
  "en" : { "nonterm_tags" : { "grup-verb" : "claus",
                              "sn" : "sn-chunk",
                              ...
                            },
           ...
          },
  ...
}
```

A nivel de Proyecto, la única estructura disponible hasta el momento en la implementación es una instancia para cada uno de los diccionarios de lexicones indexados por idioma. A diferencia de los anteriores, llenados de información inicial durante la implementación, éstos son creados vacíos y eventualmente reflejarán preferencias del humano, ingresadas de manera interactiva, respecto a cómo valora la clasificación que nuestro sistema hace de los tipos de predicados. Esto se corresponde exactamente con la discusión del inicio de esta sección sobre los criterios para incorporar datos dinámicos a conocimiento aprendido. Si se dispone de tales criterios, tal información pensada justamente como preferencias o interpretaciones más específicas del humano se almacenan en estos diccionarios de Proyecto y no afectan lo que se considera conocimiento Estático de propósito más general y persistente.

Como puede notarse, se elige como estructura principal al diccionario ya que permite una buena jerarquización de los datos, y una fácil extensión de sus campos. Por ejemplo, al diccionario de tipos de predicados pueden agregarse más claves de tipos que se puedan considerar, así como también son fácilmente extensibles los diccionarios de lexicones con más categorías determinantes.

Gracias a una funcionalidad del lenguaje de implementación, estos diccionarios son fácilmente conservados en disco mediante volcado de su contenido a respectivos archivos, es decir usando una representación de objetos persistentes.

5.3.2. Estructuras *Dinámicas*

Considerando un paralelismo con la representación por niveles de las estructuras estáticas, todo el conocimiento Dinámico que presentamos es dependiente del Caso de uso. Su existencia en el sistema es temporal, durante la ejecución de una interpretación de caso de uso, aunque claramente y como se discutió antes, parte de esta información se puede volver persistente siendo almacenado en las estructuras estáticas.

Atendiendo específicamente al conocimiento obetenido directamente por respuestas del humano en interacción, el mismo es utilizado *on the fly* para principalmente construir nuevas especificaciones según las ambigüedades extraídass y volver a entrar al Preprocesamiento, todo esto con el enfoque de la arquitectura iterativa. Además, posibles preferencias o valoraciones sobre, por ejemplo, las clasificaciones de tipos de predicados generadas por el sistema, serían seleccionadas y almacenadas directamente en las estructuras estáticas antes desarrolladas.

Disponemos entonces de las siguientes estructuras *Dinámicas*. Las mismas almacenan datos provenientes de la ejecución del sistema sobre una especificación particular de entrada. Los datos no provienen de interacción con el humano, ya que los cuales por ahora no son llevados a representaciones persistentes.

- Estructuras que almacenan segmentos de texto en lenguaje natural, obtenidos desde la expresión del caso de uso de entrada durante la etapa de Preprocesamiento. Existe un conjunto por cada longitud de ventana –esto es, la longitud de los segmentos a almacenar, contada en cantidad de palabras– en un rango de 2 a 5. Cada uno de estos conjuntos $SEGS_i$ tiene como elementos a pares de la forma `(segment, indices)`, donde `segment` es lista de i palabras, representando el orden lineal de las mismas en una cadena de texto de i palabras que aparece como segmento del texto de entrada, e `indices` una lista de los respectivos i índices enteros únicos en el texto original. El orden entre palabras de un mismo predicado, es decir que comparten un contexto reducido, se induce por el orden de los índices.

- Diccionarios que asocian los estados a los predicados cuyo alcance los contienen: `d_states_and_input_scopes` para asociar los estados a índices enteros únicos de predicados de input, y análogamente un diccionario para los de output.

- `d_preds`, diccionario de asociación de cada clave de índice entero a una referencia a su respectiva instancia de predicado. Este diccionario permite una decodificación casi directa de las etiquetas del grafo codificado, obteniendo la expresión regenerada en lenguaje natural mediante el acceso al atributo `string` de la instancia correspondiente de `Predicate`.

5.3.3. Fenómenos e Interacción

Ya se ha mencionado tanto la importancia central de la estrategia interactiva de la arquitectura del sistema, como la complementariedad entre esa componente de conocimiento Dinámico y las representaciones estáticas. Esta información almacenada le da a nuestro proyecto el cuerpo de un enfoque sistemático y persistente, pensado

para poder resolver una gran variedad de especificaciones y a la vez para acrecentar ese conocimiento adquirido mejorando su rendimiento.

Esta complemento entre estático y dinámico debe ser entendido con cuidado, ya que se trata de incorporar datos desde el exterior del sistema. Como ya se explicara, la decisión desde la información provista por el humano es inicialmente *on the fly*, en el sentido de que su utilización es inmediata y sin tener que llevarse a una estructura de datos. Es apropiada para extraer correctamente las ambigüedades presentes en ese momento, eventualmente regenerando un caso de uso similar al de entrada pero con las ambigüedades detectadas. Pero en esta implementación, no se requiere conservar esta información interactiva para futuras interpretaciones.

El desafío está entonces en la incorporación de parte de esa información a una representación permanente como conocimiento Estático. Por lo tanto, se debe identificar cuáles de los elementos de información resultantes de una interacción con el humano se deberían almacenar de manera persistente. Estos elementos consisten, en general, en unidades de información diferentes a las que se están utilizando. Concretamente, si una respuesta del humano es útil para extraer una ambigüedad, la expresión que se extrae no fue provista por el humano, sino que se conocía como posible solución y se consultó por la opción que más se adapta a lo que se quería especificar. Entonces, lo que puede almacenarse de esa interacción es el incremento en uno de un contador de la cantidad de veces que cada opción se prefirió. Es decir, la expresión como unidad de información induce el almancenamiento de un valor de contador como consecuencia de la interacción. Aún no están implementadas estas estructuras de datos y su lógica de proceso, pero de incorporar estas modificaciones de diseño en la instanciación, en el futuro el sistema puede recurrir a este conocimiento para tomar decisiones de interpretación.

Otro detalle de las estrategias de a llevar información interactiva al conocimiento Estático se relaciona con el nivel al que se incorporen. Por ejemplo, una posible interacción solicita al humano que valore por sí o no cada uno de los tipos de predicados que se identificaron. Más allá de las modificaciones a nivel de Caso de uso sobre el conocimiento Dinámico, pueden modificarse los lexicones de tipos agregando o cambiando apropiadamente los verbos y sustantivos núcleos presentes en el predicado valorado por el humano, según el tipo que ha sugerido. Este tipo de información debería reflejarse en estructuras a lo sumo a nivel de Proyecto, ya que no esperaríamos que ciertas sugerencias a priori sean tan determinantes para que se modifique el conocimiento más estable y preciso a nivel de Sistema. Para esto, propondremos algunas mejoras a la estrategia baseline de sinónimos de la subsección 5.1.2.

Hemos sugerido en el capítulo de Arquitectura del Sistema el formato de las preguntas para el humano en interacción. Claramente, no se trata del formato textual

de la pregunta sino del nivel de lenguaje de la misma. Tanto en la interacción para Extracción como para Interpretación, si bien se intentan resolver fenómenos distintos en cada una, el nivel de las preguntas es el mismo: *¿qué quiso expresar con...?*. Nada se le informa al humano de ambigüedades, de su detección temprana, de expresiones y patrones. Y además la respuesta está determinada en un conjunto pequeño de opciones. Es muy importante durante la instanciación del sistema dedicar el tiempo apropiado a reducir la complejidad de una pregunta. Con esto se reduce no sólo el impacto de comprensión de la misma sobre el humano, sino también la dispersión que vendría de un espectro amplio de respuestas posibles, en particular, y como caso extremo, si se permitiera ingresar dinámicamente nuevas interpretaciones.

Nuestra implementación actual presenta las estructuras de las componentes Estática y Dinámica expuestas en las subsecciones previas. Al momento de pretender extenderla con el desarrollo de las lógicas interactivas, debe notarse que es posible y directo pensar en una independencia entre las etapas de Extracción e Interpretación. Una independencia suficiente para implementar la interacción con el humano en la detección temprana de las ambigüedades aún preservando sólo las heurísticas ad-hoc, y luego eventualmente también incluir las interacciones de la interpretación.

En resumen, se implementó una versión del sistema con las siguientes características:

- Flujo principal de la arquitectura.
- Heurísticas ad-hoc de fenómenos simples.
- Estructuras de conocimiento y regeneración.
- Fenómenos semánticos: tipos y alcances de predicados.
- Interacción para tipos de predicados.

Capítulo 6

Conclusiones y Trabajo Futuro

(Volver al Índice general)

En este trabajo, hemos estudiado el problema de la especificación de software desde el punto de vista del tratamiento de un texto en lenguaje natural.

Hemos identificado el problema de la elicitación, esto es, de obtener una especificación de requerimientos acorde con gran precisión a las ideas originales de un cliente, destacando el enorme impacto que los errores de esta etapa generan en el costo de resolverlos en otras posteriores.

Nos hemos centrado particularmente en los requerimientos funcionales, presentados a través de un conjunto de casos de uso. Cada uno de estos casos de uso constituyen un elemento abundante en expresiones en lenguaje natural. Asumiendo ciertas convenciones en el estándar de estos textos, relevamos los aspectos destacados de esta especificación, e identificamos la presencia de la ambigüedad como el problema clave en el procesamiento de los requerimientos.

Con el objetivo de diseñar un sistema que pueda interpretar automáticamente un caso de uso así expresado, estudiamos la bibliografía de sistemas cuyos objetivos involucraban algún tratamiento de requerimientos en lenguaje natural. Observamos que en general estos sistemas incluyen estructuras muy rígidas que escapan a un humano no educado en conceptos de elicitación de requisitos. Se aprecia que existen múltiples representaciones a obtener y enmarcadas en variados niveles de abstracción. Además, los sistemas trabajan sobre textos en idioma inglés. Por último, no existe en general interacción o retroalimentación.

La solución que se propone a este estado de problema es la de diseñar un sistema que intente resolver esas desventajas. Así, se establece un formato flexible para la confección de casos de uso, y se diseña una arquitectura iterativa y de enfoque *human-in-the-loop*, es decir de fuerte interacción con el humano. Se identifica la necesidad de una fuerte complementación entre fuentes de información Estáticas y Dinámicas, así como de una

simplificación en las expresiones a preguntar, para reducir el impacto en el humano y la dispersión de sus respuestas. La interpretación de las especificaciones se hará a través de una lista de heurísticas ad-hoc. La representación a obtener es la de un grafo de arcos dirigidos, decorados con predicados codificados.

Se obtiene un cuerpo de ejemplos de casos de uso de escaso volumen dadas las restricciones impuestas a su calidad, y se recopilan desde ellos numerosos detalles sobre la naturaleza de estos elementos de elicitación. Con estos datos, hemos propuesto un conjunto de fenómenos semánticos y lingüísticos a estudiar en una instanciación de la arquitectura. A su vez, también se proponen estructuras de datos para el conocimiento a adquirir.

Hemos realizado una implementación concreta del flujo principal del sistema, incorporando conocimiento estático inicial que se obtuvo de una observación de los ejemplos, y dejando para futuras extensiones la implementación de la interacción.

Podemos concluir también que se vuelve evidente la necesidad de incrementar el volumen del corpus para capturar más conocimiento estático inicial, y para observar siquiera en una evaluación anecdótica el rendimiento del sistema.

Observamos que los ejemplos estudiados forman parte de casos de uso definitivos o al menos muy revisados, lo cual implica que muy probablemente no contengan expresiones de gran ambigüedad y complejidad sintáctica. Con la intención de ayudar al humano a reconocer sus propias ambigüedades, sería más propicio disponer de un corpus de ejemplos de casos de uso parcialmente revisados, en estado de confección, aunque casi con seguridad sea aún más difícil conseguir semejante corpus.

6.1. Trabajo Futuro

Este trabajo estudia el diseño y la instanciación de un sistema con un enfoque muy particular para un problema observado. Muchas perspectivas se abren desde el estadío actual del proyecto. Las siguientes son algunas líneas de trabajo futuro que proponemos.

Modelo de Anotación en un Contexto Educativo. Uno de los cuellos de botella más importantes en el diseño de nuestro sistema es que se basa, fuertemente y ad-hoc, en conocimiento asociado a expresiones lingüísticas y el corpus disponible no es voluminoso. Se han podido observar casos de expresiones y patrones, e intuitivamente se desprenden otros similares, pero un corpus inicial de un mayor volumen daría una base de ejemplos suficiente para incorporar el conocimiento que otorgue una buena precisión en el rendimiento del sistema. Más aún, la carencia de un corpus de ejemplos etiquetados determinó la necesidad de optar

por un conjunto de estrategias distinto al del aprendizaje automático. Si se dispusiera de un enorme corpus anotado, en el horizonte se plantea la utilización de tal aprendizaje. Como bien se expresara, la intención en el diseño es que toda interpretación resulte en una anotación del ejemplo que se interactuó, esto es, el sistema incorpora esa información desde la respuesta exterior del humano no sólo para resolver el ejemplo particular sino también para engrosar su volumen de conocimiento.

Así, proponemos adquirir un volumen inicial en un contexto educativo. Concretamente, el uso de nuestro sistema por parte de los estudiantes universitarios de la carrera de Ciencias de la Computación durante la materia de Ingeniería del Software 1. Los estudiantes de esta materia realizan un proyecto en el que atraviesan todas las etapas del modelo de cascada, y durante la primera de esas etapas, la de análisis y especificación de requerimientos, deben confeccionar un documento de especificación que incluye casos de uso. Serían los primeros clientes de nuestro sistema al proveerlo de recursos para interpretar, y a través de sus interpretaciones adquirimos retroalimentación que se expresa en ejemplos anotados para las componentes de conocimiento.

Esta utilización del sistema en el contexto académico también resulta beneficiosa para los estudiantes ya que se hacen conscientes de las ambigüedades de sus expresiones, en definitiva, conscientes de la dificultad del proceso de elicitación de requerimientos. Todo esto en concordancia con una de los lineamientos de la interacción, que intenta crear hábitos de escritura de especificaciones entre los usuarios no experimentados. De esta manera se pone en práctica la estrategia *Human-in-the-loop* en cierta escala y a la manera de una masa de recursos humanos resolviendo estas tareas simples y obteniendo beneficios por su trabajo.

Claramente, el volumen que se adquiera no será suficiente para la aplicación de un aprendedor automático. Pero sí será muy útil para hacer una prueba de rendimiento de sistema en el estadío en el que se encuentre, más precisamente, una evaluación anecdótica del mismo. Y claramente, aunque el sistema tuviera deficiencias en su performance, de todas formas se adquirirá una buena cantidad de datos para incorporar al conocimiento.

Ya se hizo mención de la independencia con la que puede realizarse una implementación del módulo de Extracción que interactúe con el humano mientras el módulo de Interpretación preserva sus heurísticas ad-hoc. Luego, concretar este modelo de anotación implica concluir la implementación de las componentes interactivas y complementarlas con estructuras de datos que almacenen los ejemplos

anotados, además de contadores de eventos que sirvan para relevar cantidades de soluciones preferidas y permitan luego ponderar estadísticamente las reglas de decisión. El modelo de anotación resulta por tanto de aislar estas componentes del flujo principal considerando sus procesos y estructuras.

Extensión de las Heurísticas. El sistema mejorará –tanto a nivel de mayor cobertura como de mejor precisión– a medida que vayamos incorporando más claves al conjunto inicial. Proponemos desarrollar heurísticas específicas para casos particulares que aún no fue posible implementar porque son muy particulares, pero que en algunos casos ya se identificaron y trataron, como los fenómenos (5) y (6) de la subsección 5.2.1.

Además, posiblemente incorporando probabilidades para disponer de un criterio de confiabilidad de cada heurística que además sirva para ordenar la lista de posibles reglas para un determinado patrón. Es decir, las confiabilidades complementaría las respuestas en interacción. Estas probabilidades de confiabilidad pueden obtenerse de los contadores de eventos como parte del modelo de anotación.

Abstracción de los Patrones de Fenómenos Lingüísticos. Resulta de enorme utilidad diseñar estructuras de datos para abstraer los patrones de los fenómenos lingüísticos descritos previamente.

Hasta ahora, los fenómenos que activan la aplicación de las heurísticas son representados a través de las lógicas de procesamiento, es decir algorítmicamente mediante bloques de comandos condicionales. Si bien se han unificado las funcionalidades similares mediante factorización de código fuente, una abstracción de datos para estos patrones implica un enorme impacto en el costo de mantenimiento de las heurísticas y de la extensión de su conjunto de reglas para capturar nuevos patrones y expresiones.

Además de estos beneficios, tal abstracción es también útil para el modelo de anotación, ya que el mismo requiere de estructuras de datos fundamentales como ésta para relevar sistemáticamente el conocimiento de las interacciones.

Por último, una buena abstracción de datos para estos fenómenos provee un criterio de jerarquización de las heurísticas expresado como análogo a la jerarquía entre las estructuras de datos, entendiendo éstas, por ejemplo, como relacionadas por especializaciones y herencias en el sentido de la orientación a objetos, u otro tipo de enfoque de jerarquización.

Extensión del Conjunto de Fenómenos Semánticos. Un posible fenómeno a considerar es el de las expresiones temporales, como la del tiempo máximo de espera para una acción.

Deben tomarse decisiones que en lo posible no involucren modificar el formato de entrada del caso de uso y dejar el tratamiento de este fenómeno a nivel de expresiones con claves lingüísticas a extraer e interpretar. Además, se debe reflejar en información en el grafo objetivo.

Estrategias para los Fenómenos Estudiados. Proponemos mejorar las estrategias de tratamiento de algunos fenómenos. En el caso del tratamiento de tipos de predicados, se considera buena la estrategia baseline de lexicones asociados a cada tipo, aunque se expusieron algunos inconvenientes con la adquisición de tales lexicones. En el futuro debemos estudiar alternativas como las siguientes: interacción con el humano, aprendizaje, uso de recursos como WordNet –con WSD y heurísticas de expansión.

Para la etapa de regeneración de lenguaje natural, el trabajo futuro debe considerar el conocimiento que se obtiene del texto de entrada y llevarlo a una estructura de diccionarios suficiente en informatividad para los objetivos del módulo, pero de un volumen más reducido. Además, se pueden tratar de construir otras estructuras de representación.

Herramienta de Preprocesamiento. Respecto a la herramienta FreeLing de análisis del módulo de Preprocesamiento, una reestructuración general se obtendría de cambiar a otro analizador. Las ventajas asociadas serían las de eventuales mejoras en la calidad de los árboles de dependencias obtenidos y en sus tiempos de cómputo. Esto es, ventajas apenas perceptibles y muy complejas de alcanzar dadas la complejidad computacional inevitable de este tipo de análisis lingüístico y el hecho de tratarse de una de las mejores herramientas para nuestro idioma. Al contrario, implicaría costes de reimplementación de las componentes y lógicas asociadas.

Así como se aprovechó la inspección y la documentación de la herramienta para decidir incorporar directamente conocimiento estático en los archivos de locuciones, de manera similar se puede agregar conocimiento al que viene con la herramienta para mejorar su rendimiento. Por ejemplo, reentrenando el analizador, o modificando las gramáticas.

En el caso de la portabilidad a otros idiomas, la solución de cambio de analizador es más viable ya que, como bien se explicara, el módulo de preprocesamiento está

parametrizado por idioma. Luego, puede preservarse la herramienta FreeLing para idiomas como el español, y utilizarse otra para la entrada en otro lenguaje.

Tratamiento *Interespecificaciones* de un Proyecto. Una mejora interesante es el diseño de estrategias para dar tratamiento más inteligente a un conjunto de especificaciones de un mismo proyecto.

El proceso tomaría como dato de entrada, como ahora, a cada caso de uso, pero el conocimiento que se obtenga de un caso sería muy informativo para el tratamiento del resto. En general, los casos de uso del mismo proyecto comparten un vocabulario de denominaciones de sistemas o componentes, de agentes involucrados, y suelen usar los mismos verbos y convenciones de expresiones para estructuras de acciones similares. Por lo tanto, las desambiguaciones que se puedan resolver para alguna expresión deben propagarse no sólo al resto de las ocurrencias de la expresión en el caso de uso sino también a las demás especificaciones.

Una posible implementación de este tratamiento es por modificación de las componentes de conocimiento para representar temporalmente cierta información de consulta prioritaria al momento de resolver algunos fenómenos como el de tipo de predicados. Otro mecanismo puede involucrar un cambio mínimo en el formato del texto plano que permita indicar explícitamente el proyecto al que pertenece la especificación.

Evaluaciones. Inicialmente, se puede configurar un entorno experimental limitado para realizar una evaluación anecdótica. Una evaluación similar pero con una mayor escala y mayor interacción se obtendría durante la utilización del sistema en el contexto educativo. Una evaluación sistemática con datos cuantitativos requiere un gran volumen de ejemplos anotados y por ahora es sólo prevista en un horizonte mucho más avanzado.

Además de la precisión y la cobertura de las interpretaciones, pueden determinarse otros elementos a observar en un futuro mediante alguna técnica de evaluación. Por ejemplo, información relacionada al impacto de la interacción en el rendimiento. La eficiencia del sistema está gobernada por el orden de complejidad del análisis de dependencias morfosintácticas durante el preprocesamiento y también por los períodos de espera de respuesta humana. Al principio del trabajo, destacamos la importancia de una dinámica de interacción que optimice una mínima solicitud con máxima informatividad. Es decir, tal impacto no sólo puede referirse a tiempo de procesamiento, sino también a la carga de tareas que significa para el humano una fuerte interacción. Para ello, deberían estudiarse e implementarse mecanismos para observar esa carga de tareas.

Bibliografía

Wikipedia - Amazon mechanical turk. http://en.wikipedia.org/wiki/Amazon_Mechanical_Turk. Accedida: 2014-02-03.

Wikipedia - Crowdsourcing. http://en.wikipedia.org/wiki/Crowdsourcing. Accedida: 2014-02-03.

Wikipedia - Human-in-the-loop. http://en.wikipedia.org/wiki/Human-in-the-loop. Accedida: 2014-02-03.

Wikipedia - Use case. http://en.wikipedia.org/wiki/Use_case. Accedida: 2014-02-03.

Atserias, J., B. Casas, E. Comelles, M. González, L. Padró, and M. Padró (2006, May). Freeling 1.3: Syntactic and semantic services in an opensource nlp library. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC 2006), ELRA*. Genoa, Italy.

Bajwa, I. S., M. Lee, and B. Bordbar (2012). Translating natural language constraints to ocl. *Journal of King Saud University - Computer and Information Sciences 24, 117–128.*

Boyd, S., D. Zowghi, and A. Farroukh (2005). Measuring the expressiveness of a constrained natural language: An empirical study. In RE '05: Proceedings of the 13th IEEE International Conference on Requirements Engineering (RE'05)*, Washington, DC, USA, pp. 339–352. IEEE Computer Society.*

Carreras, X., I. Chao, L. Padró, and M. Padró (2004). Freeling: An opensource suite of language analyzers. In Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC'04)*.*

Chantree, F., B. Nuseibeh, A. de Roeck, and A. Willis (2006). Identifying nocuous ambiguities in natural language requirements. In Proceedings of the 14th IEEE International Requirements Engineering Conference*, RE '06, Washington, DC, USA, pp. 56–65. IEEE Computer Society.*

Fröhlich, P. and J. Link (2000). Automated test case generation from dynamic models. In Proceedings of the 14th European Conference on Object-Oriented Programming*, ECOOP '00, London, UK, UK, pp. 472–492. Springer-Verlag.*

Grosz, B. J. and C. L. Sidner (1986). Attention, intentions, and the structure of discourse. Comput. Linguist. 12, 175–204.

Gutiérrez, J. J. (2005). *Generación de pruebas de sistema a partir de la especificación funcional*. Universidad de Sevilla.

Kiyavitskaya, N., N. Zeni, L. Mich, and D. M. Berry (2007). Requirements for tools for ambiguity identification and measurement in natural language requirements specifications. Technical report.

Larman, C. (2004). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development* (3rd ed.). Prentice Hall.

Lei, T., F. Long, R. Barzilay, and M. C. Rinard (2013). From natural language specifications to program input parsers. In *ACL (1)*, pp. 1294–1303. The Association for Computer Linguistics.

Lluís Padró, Miquel Collado, S. R. M. L. and I. Castellón (2010, May). Freeling 2.1: Five years of opensource language processing tools. In *Proceedings of 7th Language Resources and Evaluation Conference (LREC 2010), ELRA*. La Valletta, Malta.

Mogyorodi, G. (2001). Requirement-based testing: An overview. IEEE.

Munro, R., S. Bethard, V. Kuperman, V. T. Lai, R. Melnick, C. Potts, T. Schnoebelen, and H. Tily (2010). Crowdsourcing and Language Studies: The New Generation of Linguistic Data. In *NAACL Workshop on Creating Speech and Language Data With Amazon's Mechanical Turk*.

Padró, L. (2011, December). Analizadores multilingües en freeling. In *Linguamatica*, Volume 3, n. 2, pg. 1320.

Padró, L. and E. Stanilovsky (2012, May). Freeling 3.0: Towards wider multilinguality. In *Proceedings of the Language Resources and Evaluation Conference (LREC 2012), ELRA*. Istanbul, Turkey.

Pandita, R., X. Xiao, H. Zhong, T. Xie, S. Oney, and A. Paradkar (2012). Inferring method specifications from natural language api descriptions. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, Piscataway, NJ, USA, pp. 815–825. IEEE Press.

Pretlove, J. and C. Skourup (2007). Human in the loop. ABB Review 1/2007.

Price, D., E. Riloff, J. Zachary, B. Harvey, and O. Harvey (2000). Naturaljava: A natural language interface for programming in java.

Tjong, S. F. (2008). *Avoiding Ambiguity in Requirements Specifications*. University of Nottingham.

Wang, A., C. D. V. Hoang, and M.-Y. Kan (2010). Perspectives on crowdsourcing annotations for natural language processing.

(Volver al Índice general)