XIV Conferencia Latinoamericana de Informática
17avas. Jornadas Argentinas de Informática
e Investigación Operativa
Buenos Aires, Setiembre 1988.

Una Interfase de Lenguaje Natural para Programar

Adolfo Kvitca
Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires, Argentina

> ...When I use a word -Humpty Dumpty said, in a rather scornful tone- it means just what I choose it to mean, neither more nor less...
> Lewis Caroll, Through the looking glass.

Resumen:

En este trabajo se describe una Interfase de Lenguaje Natural (ILN) que permite crear programas a partir de la información contenida en una gramática semántica que determina el subconjunto de LN apropiado para interactuar con el sistema y la aplicación.

Un postulado básico del proyecto consiste en obtener un sistema cuya principal característica resida en la sencillez de utilización y la rapidez de adaptación, no siendo necesario conocimientos especializados sobre linguística para especificar la gramática.

1. Ventajas de una Interfase de Lenguaje Natural p/Programar:

Una gran capacidad de almacenamiento y un sencillísimo juego de instrucciones, son suficientes para construir cualquier programa, incluida la simulación completa de un nuevo computador (Turing 1930) [MAN/74]. Sin embargo, la noción que un computador cualquiera pueda simular a otro cualquiera, tanto existente como futuro, puede tal vez tener importancia filosófica, pero no es la solución de todos los problemas de computación: para programar es no sólo conveniente sino necesario que el programador cuente con mecanismos de abstracción [ABE/85].

Los diferentes lenguajes de programación ofrecen múltiples maneras de especificar una computación, transformando al computador en una máquina virtual cuyas características y capacidades son determinadas por la programación. Aunque el soporte material no ha cambiado, el programador puede pensar en términos de variables y no en celdas de memoria, en archivos de datos y no en canales de entrada/salida, en fórmulas algebráicas y no en registros y sumadores...

La utilización de una ILN para programar facilita y potencia las capacidades de expresión y comprensión de programas. Despues de todo, la manera más fácil de comunicarse con una computadora es la manera en que nos comunicamos con las personas.

Este trabajo ha sido financiado por el Programa Nacional de Informática y Electrónica (0190-243/86) SECYT - Argentina

Su uso presenta además las siguientes ventajas:

- El Lenguaje Natural, por su naturaleza declarativa, es el medio adecuado para proveer la descripción de problemas facilitando su expresión en terminos cercanos a la concepción original. (Los lenguajes de programación convencionales estan orientados a los procesos, no describen el problema sólo un método para encontrar la solución). [SCH/86]
- Aunque no sepa programar, cualquier persona puede entender que hace un programa solamente leyendo el codigo. Esta característica facilita también, el desarrollo de módulos por varios programadores y el mantenimiento de programas durante el ciclo de vida de un sistema.
- Al no tener que aprender rígidos detalles de sintaxis, se hace mas sencilla la tarea de programación, sobre todo, a niños y usuarios ocasionales.
- Es posible utilizarla para comunicarse con el usuario de la aplicación, proveyendo una ILN en Sistemas de Consulta a Bases de Datos, Sistemas Expertos, Aplicaciones Educativas, etc.
- La interfase constituye una poderosa herramienta para realizar complejos programas de una manera "natural" debido a su capacidad de realizar inferencias, determinar contextos y manipular complejas estructuras de datos.
- La utilización de la terminología propia del dominio de aplicación, que incluye conceptos intermedios de alto nivel, es esencial para razonar efectivamente y comunicarse con el usuario, facilitando el tratamiento de lineas complejas de razonamiento, tanto para el usuario como para el sistema. [KVI/88]

2. Antecedentes:

La técnicas actuales de la IA permiten diseñar programas que procesan satisfactoriamente frases del usuario restringidas de alguna forma: a) el usuario está limitado en la variación estructural de sus sentencias (sintaxis restringida por una gramática artificial) o, b) en el número de cosas que pueden significar (dominios con semántica restringida). [BOL/81]

La utilización de LN fue propuesta por muchos autores [BAL/75],[GRE/77],[WOO/77] como la mejor manera para programar. Entre ellos merecen ser destacados los siguientes sistemas: El sistema SAFE [BAL/77] permite realizar especificaciones de programas que pueden estar incompletas o ambigüas, pero en una sintaxis y vocabulario limitado. TEIRESIAS [DAV/82] es un sistema que permite al usuario el ingreso y depuración de conocimiento para un sistema experto de diagnóstico médico, en un subconjunto reducido de LN. En [YEO/84], [DAH/81] se describe como traducir un subconjunto de LN a lógica procedural. Simmons [SIM/84] muestra una aplicación de ayuda a la documentación de programas, utilizando una gramática para hacer corresponder procedimientos a descripciones en LN. Heidorn [HEI/76] desarrolló un sistema que permite especificar "problemas de colas", mediante diálogos interactivos en LN. En [VAN/80] se describe un sistema para producir funciones Lisp a partir de descripciones en LN. Biermann y Ballard comparan la eficacia de programación en lenguaje natural en la manipulación de matrices, concluyendo que puede programarse más rápido y con menos errores en LN [BIE/76].

Bertram Raphael [BER/68] propuso la idea de comparar frases contra plantillas fijas, y el sistema de consulta a Bases de Datos, LIFER [HEN/76] ha retomado esta idea. El autor ha desarrollado también un sistema de consulta a base de datos que utiliza estas ideas [KVI/84].

Como se vé, estos sistemas presentan interfases limitadas. El uso del lenguaje en forma fluida, típico de los humanos, es aún elusivo y constituye un área activa de investigación en Inteligencia Artificial. [SIM/86].

3. Dificultades para la utlización de una ILN para Programar:

El término "Lenguaje Natural" es muy vago. Cada uno uno tiene una intuición de qué significa comunicarse en "Lenguaje Natural", pero es difícil hacer esta noción precisa... es difícil separar las capacidades linguísticas de otras capacidades humanas como la memoria, el razonamiento, la resolución de problemas, la formación de hipótesis, la clasificacion, el planeamiento, el aprendizaje.

Por un lado, sin estas capacidades un sistema de computación usará el lenguaje en forma diferente que un humano y parecerá que no usa realmente el lenguaje, por otro lado no podemos esperar a implementar todas estas capacidades para hacer algo en términos de comunicación en "Lenguaje Natural".

Para realizar una ILN "competente", se necesita no sólo una gramática con la correspondiente sintaxis y semántica del lenguaje, sino también, y quizás más importante, la capacidad de realizar inferencias a partir del conocimiento del dominio de discurso y sobre las intenciones y objetivos del usuario [WIN/72], [WOO/77].

Estas características determinan que realizar una ILN sea una tarea muy compleja, que requiere mucho esfuerzo, y sólo para ser aplicada, como dijimos antes, a un dominio limitado y un subconjunto limitado del lenguaje.

Este alcance limitado, determina que los usuarios de estos sistemas tengan dificultades para aprender qué constituye una entrada aceptable al sistema, pues en lugar de aprender una sintaxis relativemente simple de un lenguaje de programación claramente definido, deben aprender un voluminoso y muy detallado conjunto de reglas de qué palabras y frases pueden ser usadas y cómo pueden ser combinadas. Se argumenta además, que nadie utilizaría LN para programar, aún si existiese, pues sería demasiado verborrágico. Finalmente se sostiene que si se desea manipular información en forma precisa y eficiente en una máquina, debe ser usado un lenguaje claramente definido y no ambiguo, lo que invalida la utilización de LN. [BIE/80].

4. Propuesta para una ILN para Programar:

Teniendo en cuenta las consideraciones anteriores, se pensó que era necesario idear un mecanismo para diseñar ILN cuyas principales características sean:

- sencillez de empleo
- rapidez de adaptación a diferentes dominios
- no debe requerir conocimientos linguísticos para ser definida

aunque esto fuera en detrimento de la potencia y generalidad de la interfase.

Estas características permitirían que la interfase sea definida por el mismo programador de la aplicación, evitándose el inconveniente del aprendizaje de la sintaxis.

La rapidez de adaptación a diferentes dominios aseguraría un amplio espectro de uso y permitiría enfrentar efectivamente el problema del alcance limitado respecto al dominio y subconjunto de LN, pues cada aplicación corresponde a un reducido dominio expresable mediante un conjunto pequeño de frases de LN.

Para disminuir a un mínimo la verborragia se debía manejar distintas formas de elipsis y coordinación, además de la posibilidad de realizar inferencias, deducir información implícita y utilizar la terminología del dominio de aplicación.

5. Aspectos de Diseño:

Especificar en forma abstracta las estructuras de datos y los algoritmos de un programa requiere una notación formalizada, en la cual el significado de cada enunciado lícito este definido con precisión, sin ambigüedad. Esto no impide, sin embargo, la utilización de una sintaxis natural, que permita más libertad para expresar conceptos. [WIN/72], [WOO/70]. Un postulado básico del proyecto es que el análisis del lenguaje puede hacerse mediante transformaciones de frases que utilicen plantillas definibles por el usuario (esta hipótesis no es un intento de proponer bases psicológicas para un modelo del procesamiento del lenguaje natural en el hombre, antecedentes sobre estas ideas se pueden encontrar en [CHO/68] donde se discute los méritos del uso de sentencias primitivas).

Otra hipótesis fundamental, consiste en que el usuario sabe utilizar LN con sus reglas de concordancia y los mecanismos de elisión que le son propios, por lo tanto, no es necesario controlar estos aspectos sintácticos. Los aspectos semánticos, que determinan el significado de las expresiones utilizadas, son los importantes, razón por lo cual se utiliza una gramática semántica para expresar en forma no ambigüa este aspecto.

Finalmente se eligió el lenguaje PROLOG [KOW/79],[CLO/81] como lenguaje de implementación y lenguaje objeto al que se traducirían las frases de LN, pues permite al programador organizar y especificar lo que intenta que el programa realice en terminos cercanos a la concepción original facilitando entonces la especificación del mismo en LN.

6. Gramáticas Semánticas:

Una gramática semántica es una gramática libre de contexto donde los símbolos no terminales y las reglas de producción estan gobernados principalmente por funciones semánticas, es decir que las clases gramaticales tradicionales son reemplazadas por clases motivadas por conceptos del dominio de aplicación [RIC/83].

Las características principales de las gramáticas semánticas que determinan su utilización en este trabajo son:

- Pueden ser utilizadas inmediatemente, sin requerir una interpretación semántica "a posteriori".
- Muchas de las ambigüedades que aparecen durante un "parsing" estrictamente sintáctico pueden ser evitadas, pues las interpretaciones que no tienen "sentido" no son generadas.
- Es más fácil de construir pues no requiere conocimientos linguisticos especializados.
- La desventaja consiste en su alcance limitado y la dificultad de generalización, por lo que generalmente son extensas. Sin embargo, las utilización en dominios reducidos, como corresponde a cada programa de aplicación, reduce a un mínimo estos inconvenientes.

7. Implementación de la Capacidad de Elipsis y Coordinación:

La capacidad de realizar elipsis, es decir omitir en las frases una o más palabras y/o reemplazarlas por referentes (Ej: ¿Cuánto gana Raúl y cuánto Jorge?; Calcular la raiz cuadrada de 2 e imprimirla), y la capacidad de coordinación, que corresponde a construcciones con conjunciones (Ej: Juan y Raúl van al cine) son en general difíciles de tratar pues es necesario utilizar información sintáctica y semántica para hacerlo (Ej: la frase "a y b son rectas" es equivalente a decir "a es una recta" y "b es una recta", en cambio la frase "a y b son paralelas" no debe interpretarse como "a es una paralela" y "b es una paralela"). [DAH/86], [MEL/85], [FON/85].

El enfoque de esta propuesta consiste en utilizar un conjunto de reglas heuristicas que, si bien no cubren todos los casos, se comportan adecuadamente en la mayoría, y que permiten que las excepciones sean "reconocidas" facilmente por el usuario. Estas heuristicas estan basadas fuertemente en la hipótesis que el usuario domina LN y que utiliza adecuadamente la elisión y coordinación. De esta manera aunque se reconocen frases sintácticamente mal formadas, se supone que estas no son utilizadas por el usuario, y las bien formadas son aceptadas en forma correcta por el sistema.

Los mecanismos de elipsis y coordinación se basan en un principio común:

> Se puede omitir toda palabra que ya haya aparecido en el contexto, así como todo artículo definido y preposición.

El contexto corresponde en general a la frase, pero puede especificarse alcances mayores, es decir con respecto a frases anteriores.

Para facilitar el uso, todas las frases son "filtradas" de manera de separar palabras como "del" en "de el", "al" en "a el", etc. Además las palabras correspondientes a acciones terminadas en "lo/la" como "imprimirlo", "ponerla", etc. son separadas de este sufijo, pues se reconoce esta caracteristica mediante las definiciones de la gramática.

Todas las conjugaciones de los verbos "ser" y "estar" pueden ser usadas indistintamente e inclusive pueden ser ignoradas si no son "esperadas".

Las reglas anteriores permiten resolver satisfactoriamente casos como:

¿Cuánto gana Raul y cuánto Jorge?.
Calcular la raiz cuadrada de 2 e imprimirla.
Hay una evidencia fuertemente sugestiva de que un organismo es de la clase neisseria si proviene de un lugar normalmente esteril y tiene un tinte gram negativo y forma esferica.

que se convierten respectivamente a las siguientes cláusulas:

```
gana(Raul,X),gana(Jorge,Y).
raiz(2,X),imprimir(X).
evidencia(clase(ORGANISMO,neisseria),5):-
  lugar(ORGANISMO,LUGAR),tipo(LUGAR,esteril),
  tinte_gram(ORGANISMO,negativo),forma(ORGANISMO,coco).
```

La capacidad de coordinación para términos que aún no aparecieron se obtiene mediante un segundo principio:

En todo lugar donde se "espera" un objeto puede venir una conjunción de objetos del mismo tipo, y la frase será interpretada como correspondiente a cada objeto.

Esto permite interpretar la frase:

Juan y Raul van al cine.

como:

Juan va al cine y Raul va al cine.

Este principo no resuelve el problema planteado mediante el ejemplo de "las paralelas", pero el usuario reconoce facilmente este tipo de excepciones.

9. Descripción del Sistema:

El sistema esta compuesto por tres módulos:

1. Módulo de Definición (MD): el diseñador de la aplicación define el subconjunto de LN apropiado para una aplicación, mediante una gramática semántica. La misma esta compuesta por las declaraciones correspondientes a la estructura jerárquica de los objetos típicos del dominio, y la terminología del dominio compuesta por las relaciones y funciones que se pueden establecer en el mismo. Es interesante notar que el MD cuenta con una interfase de LN construida con el mismo sistema.

2. Módulo de Traducción (MT): es generado automáticamente a partir de la definición de la gramática y utilizado para traducir el programa de LN a PROLOG. Es posible utilizar otros lenguajes objeto, por ejemplo comandos de SQL, mediante el simple recurso de invocarlos a través de un comando en lenguaje PROLOG.

3. Módulo de Generación (MG): es generado automáticamente a partir de la definición de la gramática y utilizado para traducir el programa PROLOG a lenguaje natural, necesario para fines de depuración del programa y exhibición de trazas para explicaciones. En la generación de código no se realiza elipsis ni concordancia pues el sistema no conoce las reglas sintácticas y semánticas correspondientes, que sí conocía el usuario cuando ingresó las frases. Además, al utilizarse por ejemplo en trazas de ejecución, las frases pierden el contexto original y no sería posible mantener las características de elisión y concordancia originales.

A continuación se exhibirán las características del sistema dando ejemplos de utilización del MD y comportamiento del MT generado. El MD reconoce como variables aquellas palabras que tengan todas sus letras en mayúscula, en caso contrario serán constantes. En las frases traducidas por el MT serán reconocidas como variables, los nombres de las clases declaradas en el MD (seguidas opcionalmente por un número) y las palabras de una sola letra. La utilización de paréntesis en las plantillas denota, en todos los casos, información de uso opcional. Los comentarios serán indicados entre {}. Las definiciones ingresadas a cada módulo serán precedidas por el nombre del módulo y una flecha "<-" (Ej: "MD<-"), la salida obtenida será indicada mediante el nombre del módulo y una flecha "->" (Ej: "MT->". La lectura del resto del trabajo puede requerir conocimientos del lenguaje PROLOG [CLA/82], [STE/86].

9.1 Definición de la estructura jerárquica de los objetos:

Mediante esta información se establece la estructura de los objetos del dominio y sus relaciones jerárquicas. Con estas simples declaraciones es posible procesar las siguientes características del LN:

a) interpretar anomalías semánticas
   (corresponde a controlar que los argumentos sean del "tipo" esperado)
b) utilización de objetos estructurados
   (Por ejemplo, "un camión con capacidad 2000 kg, color azul, chapa 3820264, trayecto con origen en Buenos Aires y destino en Mar del Plata")
c) independencia del orden de los argumentos
   (la frase "un rectángulo de ancho 10 cm y alto 20mm" es interpretada en la misma forma que "un rectángulo de alto 20 mm y ancho 10 cm")
d) generación de código dependiente de los datos
   (el MT genera el código adecuado en función de los argumentos recibidos, por ejemplo interpretando adecuadamente que "mover un canguro" corresponde a "saltar" y "mover una víbora" corresponde a "reptar")
e) herencia de información y estructuras
   (la declaración de una jerarquía de clases permite "heredar" la información y las estructuras correspondientes en forma similar al mecanismo implementado mediante "frames" [MIN/75], [KVI/84], [BOW/75], pero sin la posibilidad de redefinir información pues esto impone problemas de no-monotonía)

f) valores asumidos en forma implícita y métodos de obtención de la información asociados a los atributos de los objetos.
(la declaración de objetos estructurados permite especificar los valores típicos a asumir "valores default" y los métodos de obtención de información "valores if-needed" [WIN/75]).
g) incorporar conocimiento típico del dominio
(es posible incorporar objetos, relaciones y fórmulas típicas del dominio de aplicación. Ej: información correspondiente a registros de personal, conversiones entre unidades de medidas, etc.)

Objetos simples:
La plantilla general para declarar "objetos simples" es la siguiente:

ART_INDEF OBJETO (SALIDA<-ENTRADA):-EXPRESION_PROLOG

Donde,
ART_INDEF: corresponde a un artículo indefinido (un, uno, unos, una, unas) que son reconocidos como prefijos para la declaración de objetos.
OBJETO: es el nombre que se utilizará para referenciar a la clase correspondiente.
(SALIDA<-ENTRADA): indica el "patron" que deben seguir las palabras correspondientes a la ENTRADA de la clase declarada y la correspondiente conversión a SALIDA, respetando en ambos casos las convenciones de unificación de PROLOG.
Si no se especifica, se asume que la SALIDA será idéntica a la ENTRADA y que podrá contener sólo una palabra. La variable para referenciar en la expresión prolog estará formada por el nombre de el objeto con todas las letras en mayúscula y los espacios reemplazados por guiones. En caso de especificarse solamente la ENTRADA se asumirá que la salida es idéntica. Debe tenerse en cuenta que mientras las entrada puede corresponder a una o más palabras, la salida debe representar una única estructura.
EXPRESION_PROLOG: permite expresar las condiciones de validación y conversión de la ENTRADA y SALIDA.
Para aumentar la legibilidad del programa traducido por el MT, en todos los casos que haya que generar nombres internos se utiliza el nombre del objeto. Si este tiene espacios son reemplazados por "_".

A continuación se ilustra el uso de esta plantilla mediante algunos ejemplos:

```
MD<-      un barco.
MD<-      una casa.
    (no hay restricciones para barco y casa)
MD<-      una cancion :- publicada(CANCION).
    (al no especificarse ENTRADA<-SALIDA se asume que se
    utilizará CANCION para referenciarlas. Se supone que el
    usuario define además el predicado "publicada" que
    establece las opciones válidas para CANCION)
MD<-      un color :- member(COLOR,[rojo,azul,verde]).
    (las constantes aceptables para color serán rojo,azul y
    verde)
```

```
MD<-      un número (N) :- num(N).
     {se utiliza como nombre de variable la letra N, y se
     utiliza la primitiva "num" para validación. No se realiza
     conversión alguna}
MD<-      un hombre ([X|XS] <- X|XS).
     {si se ingresa por ejemplo "Juan Ramón Perez" será
     convertido a la estructura [Juan,Ramón,Perez]}
MD<-      un sistema automatico.
     {observar que el nombre del objeto puede estar formado por
     más de una palabra, no siendo necesario utilizar "guiones"
     o algún otro mecanismo para que simular esta
     característica.
MD<-      una velocidad maxima (vel(X,Y,Z) <- X Y por Z) :-
               num(X),longitud(Y),tiempo(Z).
     {si se ingresa "10 km por hora" sera verificado que
     correspondan a unidades de longitud y tiempo. El formato de
     salida de esta información será "vel(10,km,hora)." }
MD<-      un tamaño (W <- X Y) :-
            num(X),longitud(Y),convertir_a_metros(X,Y,W).
     {si se ingresa "10 metros" será verificado y luego será
     convertido a "10". Si se ingresa "10 km" será convertido a
     "10000"}
```

La posibilidad de convertir la ENTRADA al formato deseado de SALIDA, permite independizar los procedimientos de la interfase de LN diseñada. La utilidad es impresionante, permite por ejemplo, escribir procedimientos que se independicen de las unidades de medida: al definir un proceso que dibuje figuras geométricas de un tamaño indicado se lo realiza en función de un número que corresponde a la cantidad de puntos de la pantalla. Luego se especifica la equivalencia de la unidad de medida en la interface, junto con el conocimiento de conversión de unidades de medida. La interface se encarga entonces de la conversión, en forma transparente para el procedimiento. Si se establece que 1 mm corresponde a 2 puntos, "dibujar un rectángulo de ancho 10 cm y alto 20 mm" será convertido adecuadamente, indicando que el ancho es 200 y el alto 40, inclusive, gracias a la posibilidad de generar código dependiente de los datos podrían especificarse factores de conversión distintos para el ancho y el alto.

La declaración de los objetos es utilizada para realizar el control semántico y determinar los "no-terminales" de la gramática semántica. Mediante esta última se especifican las relaciones que se establecen en el dominio.

Organización jerárquica:

Existe actualmente la convicción de que los sistemas orientados a objetos son concebidos e implementados de forma más próxima a como la mente humana percibe los sistemas de la realidad: como una interacción entre objetos distintos, cada cual poseyendo propiedades y comportamientos propios, y donde cada objeto puede tener otros objetos como componentes de su estructura [JON/87]. Por esta razón la interfase provee mecanismos de alto nivel para manipular objetos estructurados: los objetos declarados se pueden organizar jerárquicamente, estableciendo relaciones entre clases y sub-clases.

Esta información determina la posibilidad de representar generalizaciones y especializaciones al poder reunir en clases, los objetos con propiedades similares.
La plantilla para declarar objetos se extiende entonces a:

ART_INDEF OBJETO (SAL<-ENT) es un CLASE:-EXPRESION_PROLOG

Donde,
CLASE: corresponde a la clase "primitiva" denominada "objeto", o a una clase definida por el usuario. Todo objeto declarado especificando la relación "es un" se puede utilizar como clase. Cuando sea necesario distinguirlos en el texto, referenciaremos a estos últimos como "objeto-clase" y a los otros como "objeto-ordinario". Por ejemplo:

MD<- una fragata es un barco.
MD<- un nodo es un objeto.
MD<- un arco es un objeto.
MD<- una entidad es un nodo.
MD<- un archivo y un proceso son entidades.
(el término "es" puede reemplazarse por "son" para poder utilizar en forma natural la facilidad de elipsis del sistema. Notar además que pueden usarse los plurales de los términos declarados)
MD<- un flujo es un arco.

Estas declaraciones permiten que el MT, al esperar un objeto de determinado "tipo", acepte también los "sub-tipos" correspondientes. Para el ejemplo anterior, en las declaraciones donde se utilice "entidad" también son aceptados objetos de tipo "archivo" y "proceso".

Al referenciar un objeto, puede precederse en forma opcional por el nombre de su clase (Ej: "Sarmiento" o "la fragata Sarmiento"; "facturación" o "el proceso de facturación"), este nombre es utilizado por el MT para el control semántico y luego es descartado, para poder independizarse de la forma en que se ingresen los datos.
Es posible parametrizar al MG para que produzca la conversión a lenguaje natural incluyendo, o no, el nombre de la clase a que pertenecen los objetos. También puede parametrizarse el MT para que genere código que realice control de tipos en tiempo de ejecución, en forma similar a [DAH/81].

Objetos estructurados:

Como se expresó anteriormente, los objetos pueden tener otros objetos como componentes de su estructura, permitiendo una gran flexibilidad para organizar y representar los objetos del dominio y las relaciones entre ellos. Se implementó un mecanismo para representación del conocimiento mediante objetos estructurados similar al implementado mediante "frames" pero sin posibilidad de redefinición de información por los problemas de "no-monotonia" que trae aparejados. Es decir que si se estableció un valor "default" para un atributo no puede ser declarado distinto para alguna instancia de su clase.

Nuevamente generalizamos la plantilla anterior para declarar la estructura de un objeto-clase:

ART_IND OBJETO (SAL<-ENT) es un CLASE con ESTRUCTURA:
EXPRESION_PROLOG

donde,
ESTRUCTURA: corresponde a una declaracion de atributos permitidos para el OBJETO. Estos atributos deben estar a su vez declarados como objetos. En cada atributo es posible especificar en forma opcional, encerrandolos entre paréntesis, un valor "default" o un mecanismo de obtención del valor "if-needed".

Aunque en la plantilla se especifica la constante "con" para reconocer el inicio de la estructura, el MT reconoce, en forma indistinta, los términos "con", "de", "tiene" "que tiene", "cuyo", "cuya". Esta equivalencia puede alterarse para agregar o eliminar términos.

A continuación se muestran algunos ejemplos de declaración y utilización de objetos estructurados. Al utilizarlos, es importante observar que se debe nombrar el atributo y a continuación su valor. No es necesario respetar ni el orden ni la cantidad de los atributos.

```
MD<-    un medio de transporte es un objeto
        con velocidad máxima.
    {permite aceptar frases como "el vehiculo que tiene
    velocidad máxima 200 km por hora"}
MD<-    un auto es un medio de transporte
        con dueño y color y puertas y capacidad(4 personas).
    {se asume que la capacidad típica de un auto es 4 personas}
MD<-    un hombre es un objeto.
MD<-    un dueño es un hombre.
MD<-    una capacidad (X) :- num(X), X >= 0.
MD<-    una capacidad (X <- X personas) :- num(X), X >= 1.
    {la definición de capacidad muestra que es posible
    establecer más de una construcción para un objeto, que
    permite, en este caso, utilizar un número, o un número
    seguido de la palabra personas}
MD<-    una puerta es un objeto
MD<-        con modelo y cantidad.
MD<-    un modelo :- member(MODELO,[comun,especial]).
MD<-    una cantidad (X) :- num(X), X > 0.
```

Observar que "puerta" está utilizado como atributo dentro de "auto", y que "auto" es un "medio de transporte" que tiene como atributo "velocidad máxima". Por lo tanto es posible aceptar frases como "el auto de color rojo, dueño Jorge, puertas modelo especial y velocidad máxima 250 km por hora". La capacidad se asume de 4 personas. Al no especificarse la cantidad de puertas, no es posible determinarla de manera alguna. Notar que para declarar esta información no es válido escribir "cantidad de puertas 4 modelo especial" pues siempre debe especificarse primero el nombre del atributo. La manera "adecuada" sería, por ejemplo, "puertas modelo especial cuya cantidad es 4".

```
MD<-    un colectivo es un medio de transporte
MD<-        con capacidad, peso(mult(CAPACIDAD,90,PESO)).
    {La expresión entre paréntesis determina que en caso de
    requerirse el peso hay que evaluar la expresion Prolog que
    se encuentra a continuación de la barra, el resultado
    estará en la variable "PESO"}
```

Utilizando estas declaraciones el MT crea además funciones de acceso a los atributos de un objeto, que permiten frases como "la capacidad del colectivo", "el color de el auto", etc. Todas estas funciones son de la forma:

ART_DEFINIDO ATRIBUTO de ART_DEFINIDO OBJETO.

El comportamiento del MT es diferente para los objetos ordinarios y las clases:

- Objeto-Ordinario: se procede a realizar la validación correspondiente y se genera una variable para referenciarlo.
- Objeto-Clase precedido de artículo indefinido: si está en el consecuente de una regla se lo trata como un objeto ordinario, pero si está en el antecedente o en una consulta, se crea una nueva instancia del objeto.
- Objeto-Clase precedido de artículo definido: si existe una referencia a él en el contexto, se lo trata como un objeto ordinario, en caso contrario se genera el procedimiento para extraer el objeto de la base de datos del sistema.

Los siguientes ejemplos ilustran mejor estas diferencias:

```
MT<-      un auto de color verde, dueño alberto.
MT->      es_un(o1,auto). color(auto,o1,verde).
          dueño(auto,o1,alberto).
MT<-      jorge mira un auto de color rojo.
MT->      es_un(o2,auto). color(auto,o2,rojo). mira(jorge,o2).
     {se crea un auto de color rojo}.
MT<-      jorge mira el auto de color verde.
MT->      es_un(X,auto),color(auto,X,verde),mira(jorge,X).
     {se halla el auto de color verde que mira jorge}
MT<-      Quien es el dueño?
     {elipsis ¿Quien es el dueño de el auto de color verde?}
MT->      es_un(X,auto),color(auto,X,verde),dueño(X,Y).
MT<-      dibujar un auto de color rojo y uno de color azul y
          movilizar el de color rojo.
MT->      es_un(o3,auto). color(auto,o3,rojo).
          es_un(o4,auto). color(auto,o4,azul).
          dibujar(o3),dibujar(o4),movilizar(o3).
     {la referencia al auto de color rojo corresponde al ultimo
     nombrado}
MT->      dibujar el objeto de color rojo.
MT<-      es_un(X,Y),color(Y,X,rojo),dibujar(X).
     {objeto referencia a cualquier objeto}
```

El MT acepta también que el identificador de los objetos sea establecido por el usuario mediante la plantilla:

IDENTIFICADOR es ART_INDEFINIDO OBJETO

Observar sin embargo, que esto no define nuevas clases, solamente corresponden a identificadores para instancias del programa:

```
MT<-      margarita es un barco.
MT->      es_un(margarita,barco).
MT<-      mario, alberto martinez y raul son hombres.
MT->      es_un([mario],hombre).
          es_un([alberto,martinez],hombre).
          es_un([raul],hombre).
```

```
MT<-      k27 es un sistema automatico.
MT->      es_un(k27,sistema_automatico).
```

9.2 Definición de Terminología:

Una vez declarados los objetos del dominio, es necesario declarar la terminología del dominio, compuesta por las relaciones y funciones aplicables a los mismos. El sistema realiza una traducción automática a PROLOG de estas relaciones y funciones, sin embargo, el usuario puede especificar su propia traducción.

La plantilla general corresponde al siguiente esquema:

FRASE (RESULTADO) / TRADUCCION:-EXPRESION_PROLOG.

donde,
FRASE: indica la forma esperada de la frase. En la misma se deben utilizar las referencias a los objetos declarados anteriormente.
RESULTADO: en caso de indicarse, corresponde al nombre de la variable donde se devuelve el resultado de la evaluación de la TRADUCCION. Notar que la FRASE actúa como una función que retorna un valor.
TRADUCCION: refiere al formato interno al que debe ser traducida la FRASE. Esta traducción es opcional, y puede ser generada automáticamente por el sistema con las siguientes convenciones: el nombre del predicado estará formado por las constantes de la frase, las variables corresponderán a los objetos de la frase en el orden en que aparecen.
EXPRESION_PROLOG: expresa las condiciones en las que se debe efectuar la traducción.

Las siguientes definiciones (MD) permiten que el MT traduzca adecuadamente frases como la siguiente:

```
MT<-      un proceso consulta a un archivo <-
          un flujo esta destinado al proceso y
          esta originado por el archivo.
MT->      consulta_a(PROCESO,ARCHIVO):-
          destinado(FLUJO,PROCESO),originado(FLUJO,ARCHIVO).

MD<-      un flujo esta destinado a una entidad /
          destinado(FLUJO,ENTIDAD).
MD<-      un flujo esta originado por una entidad /
          originado(FLUJO,ENTIDAD).
MD<-      un proceso consulta a un archivo.
     (la traducción generada es "consulta_a(PROCESO,ARCHIVO)")
```

En este ejemplo se puede observar el uso de elipsis en la palabra flujo y la utilización de articulos definidos para referenciar a proceso y archivo. Como se explicó, los objetos declarados son reconocidos como variables, generándose nombres adecuados. El mecanismo de validación semántica reconoce que proceso y archivo son entidades.

En el siguiente ejemplo se puede observar un uso más complejo de elipsis:

```
MT<-    Un bloque de color verde esta sobre una mesa y uno
        azul esta sobre el verde y uno rojo sobre el azul.
MD<-    un bloque es un objeto con color.
MD<-    un apoyo es un objeto.
MD<-    un bloque es un apoyo.
MD<-    un bloque esta sobre un apoyo.

MT->    es_un(o5,bloque).   color(o5,verde).   es_un(o6,mesa).
        esta_sobre(o5,o6).
        es_un(o7,bloque).   color(o7,azul).
        esta_sobre(o7,o5).
        es_un(o8,bloque).   color(o7,rojo).
        esta_sobre(o8,o7).
```

A continuación se muestra una interfase para un sistema experto en diagnóstico médico tipo MYCIN [DAV/82].

```
MD<-    un organismo.
MD<-    un lugar.
MD<-    un tipo de lugar.
MD<-    un tipo de lugar (X <- normalmente X).
MD<-    un tinte gram :-member(TINTE_GRAM,[positivo,negativo]).
MD<-    una forma (coco <- esferica).
MD<-    una forma (bacilo <- alargada).
MD<-    una evidencia (5 <- fuertemente sugestiva).
MD<-    una evidencia (3 <- sugestiva).
MD<-    una evidencia (1 <- debilmente sugestiva).
MD<-    una clase.
MD<-    un organismo proviene de un lugar /
            lugar(ORGANISMO,LUGAR).
MD<-    un organismo proviene de un lugar un tipo de lugar/
            lugar(ORGANISMO,LUGAR),tipo(LUGAR,TIPO_DE_LUGAR).
MD<-    un organismo tiene un tinte gram /
            tinte_gram(ORGANISMO,TINTE_GRAM).
MD<-    un organismo tiene una forma / forma(ORGANISMO,FORMA).
MD<-    hay una evidencia de que un organismo es de una clase /
            evidencia(clase(ORGANISMO,CLASE),EVIDENCIA).
```

que permite expresar en forma "clara", reglas de gran complejidad:

```
MT<-    Hay una evidencia fuertemente sugestiva de que un
        organismo es de la clase neisseria <- proviene de un
        lugar normalmente esteril y tiene un tinte gram
        negativo y forma esferica.
MT->    evidencia(clase(ORGANISMO,neisseria),5):-
        lugar(ORGANISMO,LUGAR),tipo(LUGAR,esteril),
        tinte_gram(ORGANISMO,negativo),forma(ORGANISMO,coco).
MT<-    Hay una evidencia sugestiva de que un organismo es
        contaminante <- proviene de la traquea y tiene forma
        alargada.
MT->    evidencia(clase(ORGANISMO,contaminante),3):-
        lugar(ORGANISMO,traque),forma(ORGANISMO,bacilo).
```

Un sistema de consulta y actualización de una base de datos sobre personal podría especificarse mediante las siguiente gramática:

```
MD<-      un empleado ([X|Xs] <- X|XS) es un objeto con
          sección y sueldo.
MD<-      un sueldo (Z <- X Y) :- num(X),moneda(X,Y,Z).
MD<-      moneda(X,australes,X).
MD<-      moneda(X,W,Y) :- cambio(W,Z), Y is X * Z.
MD<-      una sección.
MD<-      un empleado gana un importe/sueldo(EMPLEADO,IMPORTE).
MD<-      un empleado cobra un importe/sueldo(EMPLEADO,IMPORTE).
     (también podría haberse escrito las dos últimas
     definiciones mediante: un empleado X un importe /
     sueldo(EMPLEADO,IMPORTE) :- X = gana ; X = cobra.)
MD<-      Cuanto gana un empleado?/
          sueldo(EMPLEADO,SDO),
          write(EMPLEADO,gana,SDO,australes).
MD<-      imprimir un objeto / write(OBJETO).
MD<-      el sueldo de un empleado (un sueldo) /
          sueldo(EMPLEADO,SUELDO).
```

Esta gramática permite consultar y actualizar la base de datos en forma interactiva, como se muestra a continuación:

```
MT<-      Jorge Perez es un empleado cuya sección es pintura y
          sueldo 100 australes.
     (se utilizó "cuya" para la estructura y se "agregó" la
     palabra "es" para aumentar la legibilidad (recordar que
     esto es permitido))
MT->      es_un([Jorge,Perez],empleado).
     seccion(empleado,[Jorge,Perez],pintura).
     sueldo(empleado,[Jorge,Perez],100).
MT<-      Raul Gonzalez es un empleado de la sección
          calibración.
     (se utilizó "de" para la estructura y se "agregó" la
     palabra "la" para aumentar la legibilidad)
MT->      es_un([Raul,Gonzalez],empleado).
     seccion(empleado,[Raul,Gonzalez],calibracion).
MT<-      Raul Gonzalez cobra 200 dolares.
MT->      sueldo(empleado,[Raul,Gonzalez],1200).
     (observar que en la definición de "cobra" se utilizó una
     traducción compatible con la generada por el MT para los
     objeto-clase)
MT<-      ¿Cuanto gana el empleado de la sección pintura?.
MT->      es-un(X,empleado),sección(empleado,X,pintura),
          gana(empleado,X,Y),write(X,gana,Y,australes).
  ->      Jorge Perez gana 100 australes.
MT<-      imprimir el sueldo de Jorge Perez.
MT->      sueldo(empleado,[Jorge,Perez],X),write(X).
  ->      100.
```

En el próximo ejemplo se puede observar que la generación de código en función de los datos es sencilla de realizar, en el mismo se define la respuesta que debe dar cada animal a el saludo "hola":

```
MD<-  un perro :- es_un(PERRO,perro).
MD<-  un pato :- es_un(PATO,pato).
MD<-  Hola un perro () / ladrar(PERRO).
MD<-  Hola un pato () / graznar(PATO).
MD<-  ladrar(X) :- write(guau!).
MD<-  graznar(X) :- write(cua-cua!).
```

```
MT<-  Mendieta es un perro.
MT->  es-un(Mendieta,perro).
MT<-  Donald es un pato.
MT->  es-un(Donald,pato).
MT<-  Hola Mendieta.
MT->  guau!
MT<-  Hdla Donald.
MT->  cua-cua!
```

Una de las características más atrayentes de la utilización de funciones, es que no es necesario la introducción de variables intermedias como pasa en PROLOG. Por ejemplo:

```
MD<-    un hombre es el tio de un hombre_2 /
            tio(HOMBRE,HOMBRE_2).
MD<-    un hombre es el hermano de un hombre_2. /
            hermano(HOMBRE,HOMBRE_2).
MD<-    el progenitor de un hombre (un hombre-2) /
            progenitor(HOMBRE-2,HOMBRE).
MT<-    un hombre es el tio de un hombre_2 <-
            es el hermano del progenitor del hombre_2.
        (observar utilización de elipsis y de la función
        esposo).
MT->    tio(HOMBRE_1,HOMBRE_2):-
            hermano(HOMBRE_1,X),progenitor(X,HOMBRE_2).
```

En muchas oportunidades, existen predicados que pueden usarse como relaciones y como funciones. En el caso anterior se podría haber definido también la relacion "un hombre es el progenitor de un hombre_2" o la función "tio". Para estos casos se provee la siguiente facilidad: Si el nombre de la variable a devolver corresponde a uno de los objetos utilizados en la frase, se define automaticamente además de la relación, una función cuyo nombre corresponde a las constantes de la relación a partir del verbo ser o estar:

```
MD<-    un hombre es el progenitor de un hombre_2 (HOMBRE).
        (define la relación "es_el_progenitor_de" y la función
        "el progenitor de").
MD<-    un hombre es el hermano de un hombre_2 (HOMBRE).
MD<-    un hombre es el tio de un hombre_2 (HOMBRE).
MD<-    un hombre es el tio abuelo de un hombre_2.
MT<-    un hombre es el tio de un hombre_2 <-
            es el hermano del progenitor del hombre_2.
MT->    es_el_tio_de(HOMBRE_1,HOMBRE_2):-
            es_el_hermano_de(HOMBRE_1,X),
            es_el_progenitor_de(X,HOMBRE_2).
MT<-    un hombre es el tio abuelo de un hombre_2 <-
            es el tio del progenitor del hombre_2.
MT->    es_el_tio_abuelo_de(HOMBRE,HOMBRE_2):-
            es_el_tio_de(HOMBRE_1,X),
            es_el_progenitor_de(X,HOMBRE_2).
MT<-    un hombre es el tio abuelo de un hombre_2 <-
          es el hermano del progenitor del progenitor del
          hombre-2
MT->    es_el_tio_abuelo_de(HOMBRE,HOMBRE_2):-
          es_el_tio_de(HOMBRE_1,X),es_el_progenitor_de(X,Y),
          es_el_progenitor_de(Y,HOMBRE_2).
```

(Se puede ver que las funciones pueden anidarse varias veces sin necesidad de introducir variables intermedias)

También podrían ser definidas funciones matemáticas, pero en la versión actual no hay forma de declarar precedencia de operadores ni utilizar paréntesis para indicarla. Sin embargo si esto no es problema puede utilizarse la interfase para manipularlas:

```
MD<-    el factorial de un numero es un numero_2 /
            fact(NUMERO,NUMERO_2).
MD<-    un numero_1 * un numero_2 (un numero) /
            mult(NUMERO_1,NUMERO_2,NUMERO).
MD<-    un numero_1 - un numero_2 (Z) /
            sum(NUMERO,NUMERO_2,NUMERO_1).

MT<-    el factorial de #X es #X * el factorial de #X - 1.
MT->    fact(X,Y):-sum(Z,1,X),fact(Z,W),mult(X,W,Y).
```

Los objetos y la terminología del dominio de aplicación constituye una fuente importante de conocimiento que es utilizada para entender las intenciones del usuario, como se puede ver en el siguiente ejemplo:

```
MT<-    enviar cartas a los clientes atrasados en los pagos.
MT->    atrasado(X),enviar_carta(X).
MD<-    un cliente.
MD<-    un periodo (X <- X dias).
MD<-    un periodo (X <- Y meses) :- mult(30,X,Y).
MD<-    un periodo es una cantidad.
MD<-    enviar carta a un cliente / enviar_carta(CLIENTE).
MD<-    un cliente esta atrasado en los pagos (CLIENTE) /
            atrasado(CLIENTE).
MD<-    la antigüedad de la deuda de un cliente
            es un periodo (PERIODO)
antigüedad(CLIENTE,PERIODO).
MD<-    una cantidad es mayor a una cantidad_2 /
            >(CANTIDAD,CANTIDAD_2).
MT<-    un cliente esta atrasado en los pagos <-
            la antigüedad de la deuda es mayor a 2 meses.
MT->    atrasado(CLIENTE) :-
            antigüedad(CLIENTE,PERIODO),mayor(PERIODO,60).
```

En el ámbito educativo se podría lograr una rápida inserción debido a la naturalidad de programación. El lenguaje LOGO [PAP/80], ampliamente difundido en las escuelas, no es más que una adaptación del lenguaje LISP [WIN/81] a fin de lograr una sintaxis mas sencilla. Sin embargo, mientras que mediante la interfase es posible escribir "dibujar un rectángulo de ancho 5 cm y alto 18 mm" en LOGO hay que hacerlo mediante la criptográfica instrucción RECTANGULO 18 50. La utilización de la ILN permitiría además utilizar un lenguaje capaz de realizar inferencias y razonar sobre los objetos que manipula! [KOW/84].

9. Derivación del MT y el MG:

Debido a la extensión del trabajo, no se describe como se deriva automáticamente el MT y el MG a partir de la información contenida en la gramática. El mismo puede ser consultado en el informe respectivo en la oficina del Programa Nacional de Informática y Electrónica.

11. Limitaciones y futuras extensiones:

Es por todos conocido los beneficios de contar con potentes diagnósticos y mecanismos de recuperación de errores, sin embargo, la mayoría de los sistemas de LN existentes sólo cuentan con un rudimentario diagnóstico de errores. Al contemplar la posibilidad de implementar estas características en nuestro sistema se presentaron graves problemas: al utilizar una estrategia de "parsing" en paralelo era posible diagnosticar "inteligentemente" los errores de las frases y proveer recuperación de los mismos, pero no era posible extenderlo para implementar elipsis, concordancia y generación de código en función de los datos; al utilizar una estrategia de análisis en profundidad se lograron las características exhibidas en el trabajo pero el diagnóstico se limita a aceptar o rechazar las frases del usuario. Se optó por una interfase más general, sin embargo, esta limitación merece ser estudiada con mayor dedicación.

El sistema fue diseñado con el objetivo principal de producir una interfase sencilla de diseñar aunque fuera en detrimento de la potencia y generalidad de la interfase. Sin embargo se puede observar las siguientes limitaciones de la versión actual que podrían ser corregidas en versiones posteriores:

- Manejo adecuado de conjugación de verbos y concordancia de género y número, mediante reglas generales y un diccionario de términos más usados.
- El mecanismo implementado para la representación de objetos estructurados es muy general, pero no es eficiente. Debería poder parametrizarse distintas formas de representación.
- La utilización de los atributos de los objetos no es del todo natural. Una solución consistiría en declarar el formato esperado para cada atributo.
- Solo se maneja coordinación de conjunciones, es necesario extenderla para manejar disyunciones.

12. Conclusiones:

El presente trabajo describe un sistema para diseñar ILN realizado con un enfoque no tradicional: en vez de buscar potencia y generalidad se ha puesto énfasis en la sencillez de empleo, la rapidez de adaptación a diferentes dominios y el no requerir conocimientos linguísticos para ser definida.

Las gramáticas semánticas resultan de suma utilidad, por su sencillez y potencia para capturar la terminología del dominio: una versión simplificada de esta interfase se utilizó en el proyecto micro-Ethos en una herramienta de desarrollo de software. La misma permite comunicarse con el usuario utilizando la terminología del dominio de aplicación y facilitó el procedimiento de formalización del conocimiento proveyendo una metodología para la captura del conocimiento [KVI/88].

Teniendo en cuenta la simplicidad del sistema, la variedad de frases aceptadas es importante, exhibiendo características avanzadas como especificación declarativa de algoritmos, validación semántica, elipsis y concordancia, manejo de objetos estructurados, capacidad de realizar inferencias, expresión natural de relaciones y funciones, utilización de información implícita, etc.

13. Bibliografia:

[ABE/85] Abelson H,Sussman G. Structure and Interpretation of Computer Programs. MIT Press.

[BAL/75] Balzer R. A Global View of Automatic Programming. Proc.3rd IJCAI

[BAL/77] Balzer R. Goldman N. Wile D. Informality in program specification. Proc.5th IJCAI,pp 389-397.

[BER/68] Bertram Raphael. SIR, Semantic Information Processing

[BAR/81] Barr A, Cohen A, Feigenbaum E, Eds. The Handbook of Artificial Intelligence. M.Kaufmann.

[BIE/76] Biermann A. Aproaches to Automatic Programming. Advances in Computers, Volume 15. Academic Press, NY.

[BIE/80] Biermann A, Ballard B. Toward Natural Language Computation. AJCL volume 6-2.

[BOL/81] Bolc L. Natural Language comunication with computers.

[BON/79] Bonnet A. Strategies for Understanding Structured English. Memo HPP-79-25.Stanford University.

[BOW/75] Bobrow D, Collins A. Ed. Representation and Understanding. Academic Press, New York.

[CLA/82] Clark K,McCabe G. Programming in Logic. Prentice Hall.

[CHO/68] Chomsky N. On the notion of 'Rule of Grammar.

[CLO/81] Cloksin W,Mellish C. Programming in Prolog. Springer.

[DAH/81] Dahl V. Translating Spanish into logic through logic. AJCL, vol 13, pp 149-164.

[DAH/86] Dahl V.,McCord M. Treating Coordination in Logic Grammars. AJCL.

[DAV/82] Davis R. Teiresias: Applications of Meta-Level Knowledge in Knowledege-Bases Systems in Artificial Intelligence. McGraw-Hill, NY.

[FON/85] Fong S, Berwick R. New Approaches to Parsing Conjunctions Using Prolog.

[GRE/77] Green C. A Summary of the PSI Program Synthesis System. Proc.5th IJCAI.

[HEN/76] Hendrix G. The LIFER manual:a guide to building practical NL interfaces.TR 138.Menlo park

[HEI/76] Heidorn G. NLPQ proyect: Natural Language for Programing Queuing Simulation. Handbook of Artificial Intelligence, Academic Press.

[JON/87] Jonathan Miguel. Orientacao para Objetos em SMALLTALK. I Simposio Brasileiro de Engenharia de Software. RJ

[KOW/79] Kowalski R., Logic for Problem Solving. Elsevier.

[KOW/84] Kowalski R. Logic as a Computer Language for Children.

[KVI/84] Kvitca A. En colaboración. Un lenguaje de consultas a bases de datos con interfase de Lenguaje Natural.JAIIO/84, Argentina.

[KVI/84] Kvitca A. Bases de conocimiento. Una propuesta de unificacion. Anales de JAIIO/84, Argentina.

[KVI/88] Kvitca A. En colaboración: Proyecto ETHOS. Apendice Informe uETHOS. FABI. Argentina.

[MAN/74] Manna Z. Mathematycal Theory of Computation. McGraw Hill, NY.

[MEL/85] Mellish C. Computer Interpretation of Natural Language Descriptions. Ellis Horwood.

[MIN/75] Minsky M. A framework for Representing Knowledge. The Psychology of Computer Vision. McGraw Hill, NY.

[PAP/80] Papert S. Desafio a la Mente, Ed.Galapagos, Argentina

[PER/80] Pereira F,Warren D. Definite clause grammars for language analysys — a survey of the formalism and a comparison with transition networks. Artificial Intelligence, vol 13. pp 231-278.

[RIC/83] Rich E. Artificial Intelligence. McGraw Hill
[ROB/80] Robinson J. DIAGRAM: A Grammar for Dialogues. TR 205, SRI International, Menlo Park.
[SCH/86] Schor Marshall. Declarative Knowledge Programming: Better than Procedural?. IEEE EXPERT, Spring 1986.
[STE/86] Sterling L, Shapiro E. The Art of Prolog. MIT Press.
[SIM/84] Simmons R. Computations from the English. Prentice Hall. New York.
[SIM/86] Simmons R. Man-Machine Interfaces: Can They Guess What You Want?. IEEE EXPERT. Spring 1986.
[VAN/80] Van Baalen J. Using Natural Language to Solve Recursive Programming Problems.
[YEO/84] Yeong Yu. Translating Horn Clauses from English.
[WIN/72] Winograd T. Understanding Natural Language. Ac Press.
[WIN/75] Winograd T. Frame Representations and the Declarative Procedural Controversy,in Representations and Understanding. Academic Press, New York.
[WIN/81] Winston P. Horn B. LISP. Addison-Wesley.
[WOO/70] Woods W. Transition Network Grammars for Natural Language Analysis. Comm. ACM. volume 13-10.pp341-398.
[WOO/77] Woods W. A personal View of Natural Language Understanding", SIGART Newsleter, 2/77,pp 17-20.