Mod-Logic, traductor de predicados tipo Prolog a una base de datos relacional e intérprete de consultas con plantillas diseñadas con SQL

María de G. Cota O., Pedro Flores P., Ivan A. López S., Melina Morales C.
Departamento de Matemáticas
Universidad de Sonora
Hermosillo, Sonora, CP 83000, México

RESUMEN

La programación lógica es importante en el desarrollo de sistemas del área de Inteligencia Artificial, y uno de los lenguajes más eficientes y utilizado en este paradigma es Prolog. Algunos compiladores de este tipo de programación hacen uso de la funcionalidad del lenguaje lógico para bases de datos deductivas y logran la implementación de consultas recursivas sobre bases de datos relacionales, sin embargo, presentan problemas de compatibilidad al momento de reutilizar el código diseñado en aplicaciones desarrolladas con otro tipo de compiladores, ya que las librerías dinámicas generadas por los mismos, incluyen operaciones básicas para almacenamiento y recuperación de información de bases de datos pero no tienen funciones que permitan construir e identificar predicados tipo Prolog en forma directa de la base de datos, encontrándonos con restricciones para diseñar y desarrollar objetos de sistemas de software que utilizan módulos lógicos independientes. Para resolver este problema, se ha desarrollado un módulo lógico (Mod-Logic) que traduce predicados tipo Prolog a una base de datos relacional, y los interpreta a través de consultas diseñadas con SQL, y una interfaz gráfica para introducir y recuperar información sin necesidad de que el usuario sea experto en programación lógica.

Palabras Claves: Prolog, Traductor, Interprete, Representación, Conocimiento.

1. INTRODUCCION

La programación declarativa permite describir la solución de un problema, y esto se logra a través de mecanismos de inferencia que, a través de la abstracción de conceptos, permiten evaluar distintas opciones para obtener una conclusión a partir de una premisa [1], [5].

La Lógica es uno de los principales fundamentos matemáticos, y una base indispensable para las ciencias de la computación. La formalización del conocimiento y la automatización de las formas de razonamiento son primordiales en algunas áreas de desarrollo científico y/o tecnológico, pero es muy relevante en la de Inteligencia Artificial [1], [5].

Compiladores como Prolog, SWI-Prolog, Visual Prolog, SICStusProlog [2], [3], [4], etc, son herramientas indispensables para la implementación de lenguajes lógicos, y permiten crear librerías de enlace dinámico, que muchas veces no son totalmente compatibles con los estándares que se manejan en otro tipo de lenguajes de programación utilizados para el desarrollo de aplicaciones, donde se hace necesario tener un esquema que permita incluir módulos lógicos como componentes adicionaes.

Datalog es un lenguaje declarativo que requiere de conocimientos avanzados en el área [5], y es considerado como una herramienta que es muy útil para la representación del conocimiento en programación lógica, utilizando bases de datos relacionales [6], lo cual proporciona funcionalidad adicional a los compiladores para este tipo de programación, en virtud de que en la formulación de predicados, permiten involucrar consultas recursivas que requieren de un análisis deductivo.

Para utilizar alternativas como las descritas anteriormente, afrontando las limitantes de utilizar solamente operaciones para almacenamiento y recuperación de información de una base de datos y los problemas de compatibilidad para enlazar los componentes lógicos con aplicaciones desarrolladas con otro tipo de compiladores, debe tomarse en cuenta que la inversión estimada tendrá costos que pueden ir desde $299.00 a $4,109.00 dólares [2], [3], [4], además de los pagos adicionales que se requieren para actualización ($149.00 dólares en adelante), sobre todo cuando no se tiene contemplado darles un uso continuo que permita recuperar la inversión.

Como una forma de solución a esta problemática, en este artículo se describe el diseño de Mod-Logic, proyecto que se desarrolla en lenguaje C/C++, con soporte para el Sistema Gestor de Base de Datos MySql, el cual brinda la posibilidad de representar la programación lógica tipo Prolog, haciendo uso de bases de datos relacionales, y a través de una interfaz amigable, permite insertar y recuperar información de la base de datos, de forma rápida y eficiente, con posibilidades de adaptarse a otros manejadores de bases de datos a futuro, utilizando los métodos que para tal efecto proporcionan cada uno de ellos.

Para efectos de organización, este artículo se divide en seis secciones: la primera corresponde a esta introducción; la segunda presenta una descripción general y las ventajas de este proyecto; la tercera contiene detalles sobre la sintaxis gramatical que fue diseñada para la programación de predicados tipo Prolog y el procedimiento de traducción de los mismos hacia una base de datos relacional; la cuarta describe la estructura de la base de datos diseñada, un ejemplo de uso y el procedimiento para la generación de plantillas utilizando SQL; y la quinta incluye las conclusiones finales de los autores de este artículo.

2. DESCRIPCION GENERAL Y VENTAJAS DE MOD-LOGIC

Según Robert Moore [1], la mayoría de las formas superiores de conducta inteligente requieren de la representación explícita del conocimiento, constituyéndose en este contexto como piedra angular, la lógica formal.

La representación del conocimiento basada en la lógica, permite contar con información sobre un evento, aún cuando no se tenga una descripción completa, lo cual brinda la posibilidad de dar respuesta a consultas complejas [1].

La principal idea de la Programación Logica está implementada en Prolog [6], lo cual lo convierte en un lenguaje idóneo para utilizarse en el desarrollo de sistemas inteligentes [1], [5].

Funcionamiento
El funcionamiento general de Mod-Logic se basa en las ideas mencionadas anteriormente con las siguientes ventajas:

a) Es una librería amigable y sencilla que puede utilizarse en aplicaciones que implementen la forma de representación del conocimiento a través de predicados tipo Prolog.
b) Cuenta con una interfaz visual y puede ser utilizada por un usuario que no tenga conocimientos avanzados en programación lógica.
c) No requiere del uso de compiladores para este tipo de programación.
d) El diseño de almacenamiento establecido evita el uso indiscriminado de tablas en la base de datos relacional.
e) Está escrita en lenguaje C/C++, lo que valida su eficiencia, rapidez y portabilidad.
f) Implementa conectividad a bases de datos relacionales utilizando el Sistema Gestor de Base de Datos MySql, sin necesidad de utilizar librerías complejas, o hacer uso de la conectividad que tienen incluidas algunos compiladores.

Además, como una línea de trabajo a futuro, se contempla la posibilidad de desarrollar o adaptar una versión de Mod-Logic para otros sistemas gestores de base de datos como Oracle, SqlServer, PostgreeSql, o a través de ODBC. Para este procedimiento bastará con utilizar las funciones que se proporcionan por cada uno de ellos para conectividad, almacenamiento y recuperación de información de las bases de datos correspondientes.

Flujo de información
El flujo de información se realiza tomando en cuenta el diseño de funcionamiento de Mod-Logic, y se lleva a cabo a través de los siguientes procedimientos:

a) El usuario puede consultar o insertar información en la base de datos a través de la interfaz visual, que a su vez tiene conexión directa con el módulo lógico, y éste con la base de datos respectiva (Ver figura 1).
b) Las aplicaciones pueden interactuar con la base de datos a través del módulo lógico, utilizando un conjunto de reglas gramáticales que fueron diseñadas para tal efecto (Ver figura 1).

Las producciones gramaticales son fáciles de implementar y se constituyen en el medio que permite aplicar los métodos de introducción y recuperación de información relacionada con predicados tipo prolog almacenados en la base de datos que para tal efecto ha sido diseñada, y que se describe posteriormente en este documento.

3. DESCRIPCIÓN GRAMATICAL

La parte gramatical ha sido diseñada para trabajar en el contexto general de otras gramáticas, por lo cual puede utilizarse con otros lenguajes fácilmente.

Este diseño se basa en la teoría de Gramáticas Independientes de Contexto (GIC), que de ahora en adelante denotaremos como G, donde consideramos que G es una cuadrupla, G = (N, T, P, S) [7] con el siguiente significado:

N Conjunto finito de símbolos No Terminales

T Conjunto finito de símbolos Terminales

S Símbolo de N, que se constituye como símbolo inicial o raíz de la GIC

P Conjunto finito y no vacío de producciones del tipo: $A \rightarrow a$, donde A pertenece a N y *a* pertenece a $(N \vee T)^+$

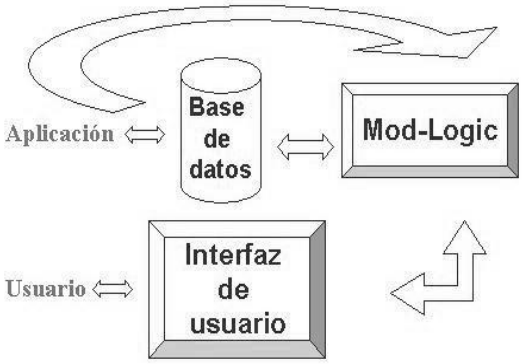

Figura 1- Flujo de información de Mod-Logic.

Para la especificación de la gramática se utiliza la Forma Normal de Backus-Naur Extendida (EBNF), que se usa para expresar la sintáxis de las GIC [8], con la notación básica que se muestra en la tabla 1.

Simbología	Descripción
|	Operador 'Or' (alternativa)
{ x }	Lista recursiva sobre elementos x
[x]	Opcionalidad sobre x
,	Concatenación 'And'
< x >	x es un símbolo no terminal
'x'	x es símbolo terminal
::=	Implicación
....	Secuencia de valores por definir
N	Máximo elemento de tabla en base de datos.

Tabla 1.- Notación básica EBNF

En la especificación de la GIC, en Mod-Lógic:
S El símbolo inicial es <representacion_conocimiento>
P Representa el conjunto de reglas.

Sintáxis de las producciones de P
La estructura general de un programa se conforma con las siguientes producciones:

< programa > ::= ‘(‘ < expresion > ‘)’
< expresion > ::= < tipo_transacción >
| < especificaciones_otros_lenguajes >
| < tipo_transacción >
<especificaciones_otros_lenguajes>
|<especificaciones_otros_lenguajes>
<tipo_transacción>
<tipo_transacción>
<especificaciones_otros_lenguajes>
<tipo_transacción>
<tipo_transacción>::=
<programación_conocimiento>
donde:
<especificaciones_otros_lenguajes>

se refiere a expresiones gramaticales definidas en las gramáticas de los lenguajes donde se incrusta la sintaxis para el manejo del módulo.

La programación del conocimiento se implementa con dos tipos de reglas:

a) Las que responden al formato de Prolog, identificadas con el símbolo de N: <regla>.
b) Las generadas por un árbol de decisión, que se representan por el símbolo de N: <condición>.

Por cuestiones de espacio, a continuación sólo se presenta la sintaxis relacionada con la parte general de programación del conocimiento, y la que se indica en el inciso a) del párrafo anterior:

<programación_conocimiento> ::= ‘insertar_regla’
‘/’ <regla> | ‘evaluar_regla’ ‘/’
<regla> | ‘buscar_regla’ ‘/’ <regla>
| ‘insertar_condicion ‘/’ <condicion>
| ‘evaluar_condicion’ ‘/’ <condicion>
| ‘buscar_condicion’ ‘/’ <condicion>
<regla>::= <cabeza> ´(´ { <argumentos> } ´)’ ´:-´
<lista_reglas> | <hechos>
<hechos>::= <cabeza> ´(´ {<argumentos>} ´)´
<cabeza>::= <letra> {<letras>}
<lista_reglas>::= <regla> [{ [´,´ | ´or´] <regla> }]
< argumentos >::= < letra > { < letra > } | < letra >
{<letra> } [´,´{<letra>}]
<evaluar_reglas>::= ´evaluar´ ´/´ ´si´ ´/´ <regla>
´/´ ´=´ ´/´ ´true´ ´/´ ´entonces´ ´/´ <accion>

Dentro de este esquema, para la generación de reglas, se cuenta con los elementos:

1) Un traductor de programas y predicados, que se encarga de interpretar y traducir los datos a una base de datos relacional.
2) Un generador de consultas SQL para interpretar la validez de la información que se filtra a través de la sintaxis del símbolo de N: <regla>, y que proviene de la aplicación o de la interfaz visual.
3) Una interfaz amigable que:
   a) Sirve como base para el diseño de predicados en forma transparente al usuario.
   b) Permite realizar consultas sobre la base de conocimientos.

Etapa de traducción
En esta etapa, Mod-Logic lee un archivo que se divide en tres secciones:

a) Semántica.- En esta sección se define el significado de los argumentos de cada predicado que se incluyen.
b) Reglas.- Esta sección incluye las reglas.
c) Hechos.- Este apartado contiene los hechos como parte de la base de conocimientos.

Flujo de datos
Como primer paso, se introduce el código que será revisado por un analizador léxico. Cuando en la etapa anterior no se encuentra ningún error, se verifica que todos los símbolos presentan un orden correcto, se realiza el análisis semántico, que consiste en revisar que cada regla y hecho está definido en la sección de la semántica y que exista coincidencia en el número de argumentos especificados. Por último, se pasa al proceso de traducción de la información hacia la base de datos (Ver figura 2.).

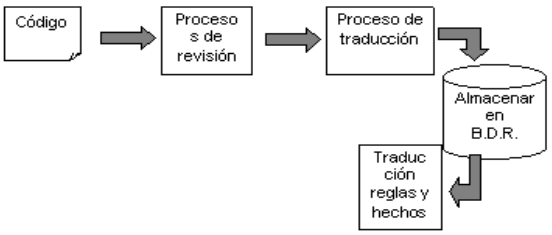

Figura 2.- Flujo de datos.

Con este procedimiento y la construcción de plantillas SQL, se asegura que los resultados que se obtengan al evaluar la información, serán iguales a los que arrojan los compiladores de Prolog

En el siguiente punto se presenta una descripción sobre el diseño de la base de datos que se utiliza en Mod-Logic..

4. ESTRUCTURA DE LA BASE DE DATOS DE MOD-LOGIC

Para guardar la información de plantillas que incluyen las referencias sobre predicados, reglas y datos que en conjunto representan el conocimiento a través de la implementación de programas lógicos, se diseñó una estructura de base de datos para Mod-Logic, que se conforma con las siguientes variables de relación (Ver figura 3.):

a) predicados.- Para identificación de la cabeza de las reglas y hechos registrados.
b) lista_reglas.- Para identificar la lista de reglas que forman el cuerpo de la regla.
c) hechos.- Guarda la información básica para conformar la base de conocimientos.

d) semántica_argumentos.- Proporciona el sentido semántico o significado de los argumentos de los hechos registrados.

e) asociación.- Permite representar las relaciones existentes entre los datos registrados en la tabla de hechos.

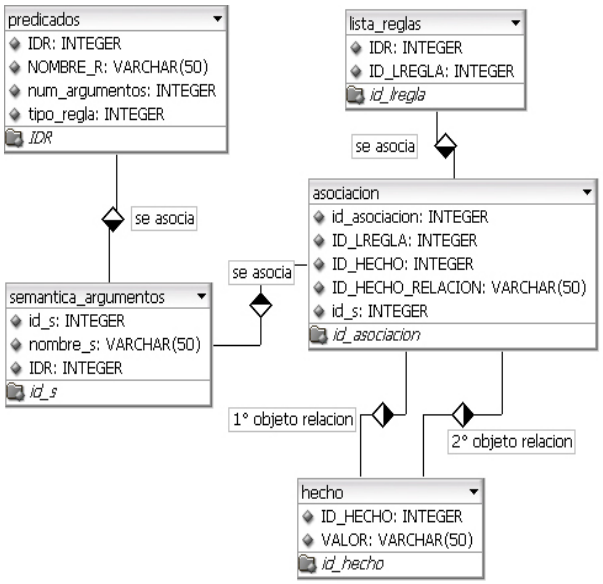

Figura 3.- Estructura de la base de datos de Mod-Logic.

En la interpretación de la información, se realiza un procedimiento para concatenar las cadenas que sirven para construir una plantilla SQL, de tal forma que se obtengan resultados iguales a los que arrojan los compiladores para lenguajes lógicos como Prolog[9].

Procedimiento para la construcción de plantillas SQL

El proceso para construir las plantillas se divide en las siguientes etapas:

a) En primer lugar se localiza el número de identificación, tipo y número de argumentos que corresponde al predicado que se desea evaluar.

b) Después, con esta información, se localiza el identificador de la semántica de los argumentos por orden ascendente, correspondiéndole:
   - En el caso de hechos: al primer argumento, se le relaciona con el campo ID_HECHO de la tabla asociacion, y al resto con el campo ID_HECHO_RELACION de la misma tabla.
   - En caso de tratarse de una lista de reglas que componen el cuerpo del predicado, se localiza el identificador del predicado principal, luego se revisa el número de reglas que componen el cuerpo del mismo, y por último, se construye la plantilla de hechos para cada una de ellas, mezclándose al final los resultados obtenidos para obtener una sola conclusión.

Para aclarar el uso de estas plantillas, a continuación se presenta un ejemplo de uso

Ejemplo de uso

Se ha elegido uno de los ejemplos clásicos para presentar este tipo de problemas, que consiste en un conjunto de relaciones familiares que existen entre un grupo de personas. Para efectos de simplificación, se utilizan nombres cortos con los siguientes valores (Ver figura 4.).

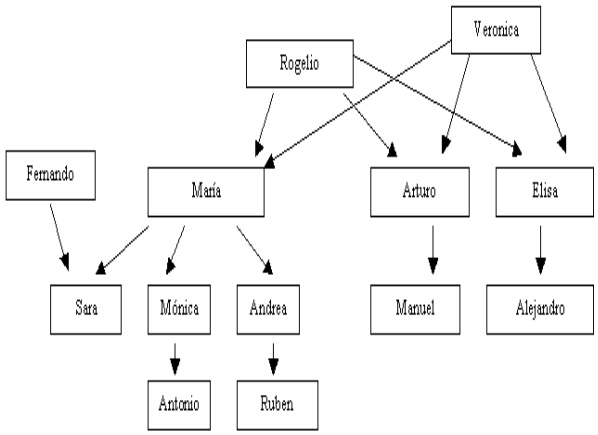

Figura 4.- Relación familiar Padre – Hijo.

Ejemplo del código a traducir:

semántica:

```
hijo(hijo,padre,madre)
padre(hijo,padre)
madre(hijo,madre)
hermanos(hijo,hijo)
progenitor(hijo,padre)
progenitor(hijo,madre)
```

reglas:

```
hermanos(A,B):-padre(P,A),madre(M,A),
        padre(P,B),madre(M,B).
progenitor(X,P):- padre(P,X).
progenitor(X,M):-  madre(M,X).
madre(M,X):- hijo(X,_,M).
padre(P,X):- hijo(X,P,_).
```

hechos:

```
hijo("Maria", "Rogelio", "Veronica"),
hijo("Arturo", "Rogelio", "Veronica"),
hijo("Elisa", "Rogelio", "Veronica"),
hijo("Sara", "Fernando", "Maria"),
hijo("Monica", "", "Maria"),
hijo("Andrea", "", "Maria"),
hijo("Antonio", "", "Monica"),
hijo("Ruben", "", "Andrea"),
hijo("Manuel", "Arturo", ""),
hijo("Alejandro", "", "Elisa"),
```

Plantilla para obtener los valores del predicado hijo:

```
Select distinct p1.nombre_r, h1.valor, h2.valor,
  h3.valor from predicados as p1 inner join hecho as
  h1 inner join hecho as h2  inner join hecho as h3
  inner join asociacion as a1 inner join asociacion as
  a2 inner join asociacion as a3 on h1.id_hecho =
  a1.id_hecho and h2.id_hecho =
  a2.id_hecho_relacion and h3.id_hecho =
  a3.id_hecho_relacion and a1.id_hecho =
```

```
a2.id_hecho  and a2.id_hecho = a3.id_hecho
and a1.id_hecho = a3.ID_HECHO
and h1.valor <> h2.valor and h2.valor <> h3.valor
and a2.id_s = 2 and a3.id_s = 3 group by h1.valor
order by a1.id_asociacion asc
```

con la cual se obtienen resultados como los que arrojaría Prolog (Ver figura 5.):

nombre_r	valor	valor	valor
hijo	MARIA	ROGELIO	VERONICA
hijo	ARTURO	ROGELIO	VERONICA
hijo	ELISA	ROGELIO	VERONICA
hijo	SARA	FERNANDO	MARIA
hijo	ANDREA	<DESCONOCIDO>	MARIA
hijo	ANTONIO	<DESCONOCIDO>	MONICA
hijo	RUBEN	<DESCONOCIDO>	ANDREA
hijo	MANUEL	ARTURO	<DESCONOCIDO>
hijo	ALEJANDRO	<DESCONOCIDO>	ELISA

Figura 5.- Resultados de consulta (Hijo-Padre-Madre).

y para la relación padre se genera la plantilla:

```
Select distinct p1.nombre_r, h1.valor, h2.valor
 from predicados as p1 inner join hecho as h1 inner
 join hecho as h2  inner join asociacion as a1
 inner join asociacion as a2 on h1.id_hecho =
 a1.id_hecho  and h2.id_hecho =
 a2.id_hecho_relacion and a1.id_hecho =
 a2.id_hecho  and h1.valor <> h2.valor
 and a2.id_s = 2  where p1.idr = 2 group by h1.valor
 order by a1.id_asociacion asc
```

con los resultados (Ver figura 6.):

nombre_r	valor	valor
padre	MARIA	ROGELIO
padre	ARTURO	ROGELIO
padre	ELISA	ROGELIO
padre	SARA	FERNANDO
padre	MONICA	<DESCONOCIDO>
padre	ANDREA	<DESCONOCIDO>
padre	ANTONIO	<DESCONOCIDO>
padre	RUBEN	<DESCONOCIDO>
padre	MANUEL	ARTURO
padre	ALEJANDRO	<DESCONOCIDO>

Figura 6.- Resultados consulta (Hijo – Padre).

Cabe mencionar que al probarse las consultas con estas plantillas, se obtuvieron resultados con rapidez y excelente calidad.

5. CONCLUSIONES

Tomando en cuenta que Prolog es uno de los lenguajes más importantes del área de Inteligencia Artificial, la cual ha cobrado gran importancia en la resolución de problemas complejos en la actualidad, y que existe una tendencia a crear sistemas híbridos con componentes escritos en distintos lenguajes de programación, es de concluir que es necesario contar con nuevas alternativas que permitan la posibilidad de agregar módulos lógicos independientes que puedan ser enlazados fácilmente con este tipo de aplicaciones.

Por otra parte, algunos de los compiladores que existen para este tipo de programación permiten la conexión con bases de datos deductivas, a través de lenguajes lógicos como Datalog, pero presentan las siguientes limitantes: este tipo de objetos pueden ocasionar problemas de compatibilidad con otros lenguajes de programación; sólo se incluyen funciones de operaciones básicas para almacenamiento y recuperación de información; no existen procedimientos que permitan guardar e interpretar la estructura de predicados del programa lógico; las licencias comerciales tienen costos muy altos, motivo por el cual, para utilizarse con este fin, es recomendable estimar que el uso que se les dará respaldará la inversión.

Este trabajo plantea una alternativa de solución para el desarrollo de módulos lógicos que se enlacen fácilmente a las aplicaciones, diversificando los lenguajes en que éstas últimas están escritas. Este tipo de módulos, permiten traducir predicados tipo Prolog a una base de datos relacional, interpretar consultas con plantillas diseñadas con SQL, y utilizar una interfaz gráfica, que en forma transparente, permite a un usuario introducir e interpretar información de la base de datos, sin necesidad de tener conocimientos avanzados de programación en el área.

Aunque Mod-Logic trabaja con la estructura de una base de datos relacional y utiliza el Sistema Gestor de Base de Datos MySql, se ha diseñado tomando en cuenta la posibilidad de ampliar su funcionalidad hacia otros sistemas gestores de bases de datos como Oracle, SQLServer, PostgreeSQL, ODBC, etc., lo cual se realizará en función de las librerías de conectividad que se proporcionan por cada uno de ellos.

6. REFERENCIAS

[1] Moore C. Robert. **Logic and Representation. CSLI Lecture Notes nº 39. CSLI Publications**, Stanford, California, 1995, pp. 1-7, 11-14.

[2] http://www.visual-prolog.com/

[3] http://www.sics.se/sicstus

[4] http://www.swi-prolog.org/

[5] Nilsson, U. & Maluszynski, **Logic Programming and Prolog**, 2000. (Disponible en http://www.ida.liu.se/~ulfni/lpp/), pp. 13-14, 103, 115, 245.

[6] Neil C. Howe. **Artificial Intelligence through Prolog by Neil C.Rowe**. Prentece-Hall, 1988, ISBN 0-13-048679-5. pp. 26, 128.

[7] Kelley Dean, **Teoría de Autómatas y Lenguajes Formales**, Prentice Hall, 1995, ISBN: 0-13-518705-2, pp. 30, 105-170

[8] V. Aho Alfred, Sethi Ravi & D. Ullman Jeffrey, **Compiladores. Principios, técnicas y herramientas**, Addison Wesley Longman (Pearson), 1998, ISBN: 968-444-333-1, pp. 25-82.

[9] Russell Stuart & Norving Peter. **Artificial Intelligence (A Modern Approach)**. Prentice-Hall, 1995, ISBN: 0-13-103805-2. pp. 304.

[10] http://www.mysql.com