Conciencia Tecnológica
ISSN: 1405-5597
contec@mail.ita.mx
Instituto Tecnológico de Aguascalientes
México

Alonso Amo, Fernando; Villalobos Abarca, Marco
Programación lógica: un enfoque para desarrollar aplicaciones
Conciencia Tecnológica, núm. 14, agosto, 2000, pp. 3-11
Instituto Tecnológico de Aguascalientes
Aguascalientes, México

PROGRAMACIÓN LÓGICA: UN ENFOQUE PARA DESARROLLAR APLICACIONES

Fernando Alonso Amo
falonso@fi.upm.es
Facultad de Informática
Universidad Politécnica de Madrid
Madrid – España

Marco Villalobos Abarca[1]
fd90157@zipi.fi.upm.es
Depto. de Computación e Informática
Universidad de Tarapacá
Arica – Chile

RESUMEN

No hay una fórmula general para un buen uso de la programación lógica en el desarrollo de aplicaciones. En este artículo se propone un enfoque metodológico para abordar esta tarea.

El enfoque metodológico, usa una notación formal para representar los conceptos necesarios y consiste en cuatro fases fundamentales: Descripción del dominio de aplicación, Diseño de una estrategia, Traducción a un lenguaje de programación que soporte el paradigma lógico, como lo es Prolog, y finalmente, una Evaluación. Se muestra su aplicabilidad para desarrollar descripciones de programas lógicos que manipulan conocimiento y resuelven problemas en base a ese conocimiento.

PALABRAS CLAVES: Programación Lógica, Enfoque de Desarrollo, Metodología de Programación.

INTRODUCCION

La programación lógica constituye una herramienta que tradicionalmente ha provisto de una sólida estructura conceptual para representar conocimiento. Puede ser usada para describir y recuperar conocimiento sobre un dominio de aplicación, para describir nuestro conocimiento sobre tareas de cálculo tradicional, y para describir tareas menos triviales como la capacidad de encontrar solución a problemas que normalmente se considera dentro del campo de la Inteligencia Artificial. Esto es porque encontrar soluciones a problemas es todavía una capacidad muy dependiente de la intuición humana, las cuales son muchas veces difíciles de desarrollar y formalizar. No hay una fórmula general para un buen uso de la programación lógica en el desarrollo de aplicaciones de este tipo, pero se intentará ilustrar en este trabajo una aproximación para abordar esta tarea (ver otros enfoques en Brewka et al. [4] y Gelfond et al. [7]). En esta aproximación se plantea un enfoque que parte desde una descripción del dominio de aplicación en términos de entidades y relaciones satisfechas por estas entidades, que a su vez permiten representar al mundo en términos de estados. Fundamentalmente, se describen un estado inicial y otro final (objetivo). A continuación, se desarrolla una estrategia que define un conjunto de acciones que transforman el dominio descrito por el estado inicial en otros estados intermedios hasta llegar al estado final. El problema se resuelve encontrando el camino que lleva desde el estado inicial al estado final. Las entidades, sus relaciones y la estrategia son traducidas a un lenguaje de programación que soporte el paradigma lógico. Y finalmente, se evalúa el producto en sus aspectos de corrección, validez y usabilidad.

CONCEPTOS PREVIOS

¿Qué es un Paradigma de Programación?

1 Doctorando Universidad de Tarapacá de Arica – Chile en la Universidad Politécnica de Madrid España.

Alonso y Segovia [1] definen "*un paradigma de programación como un modelo básico de diseño e implementación de programas...*". Ambler, Brunett y Zimmerman [2] lo definen como "*una colección de modelos conceptuales que juntos modelan el proceso de diseño y determinan la estructura de un programa*".

Ambler et al. [2] proponen que el tipo de solución que aporta un paradigma para resolver un problema particular, define tres categorías de paradigmas: Solución procedimental u operacional, Solución demostrativa y Solución declarativa. Para análisis de otra clasificación de los paradigmas ver la propuesta por Floyd [6]. Aquí interesa describir el paradigma declarativo el cual es el centro de atención de éste trabajo. La *Programación Lógica.*

¿Qué es el Paradigma de Programación Lógica?

Genesereth y Ginsberg [8] señalaron que "*la programación lógica es una programación por descripción. El programa se construye describiendo el área de aplicación, esto es, se señala el qué se desea (mediante hechos que son verdaderos) pero no el cómo obtenerlo, esto está implícito*". Asimismo, Ambler et al. [2] señalaron que "*el paradigma lógico asume la definición de un conjunto de hechos... y un conjunto de reglas que permiten la deducción de otros hechos. Así, la programación lógica, desde la perspectiva del programador, es una técnica que consiste en expresar apropiadamente todos los hechos y reglas necesarias que definen un problema*".

¿Qué es un Programa Lógico?

En consecuencia, un programa lógico se configura como un conjunto de hechos (asertos o proposiciones), y de reglas lógicas previamente establecidas, que obtienen conclusiones en base a una serie de preguntas lógicas. El control es inherente al sistema, el que permite investigar las preguntas lógicas. Esta capacidad es el concepto clave que subyace en la Programación Lógica (descriptiva). Al separar el control y la lógica, el programa lógico se transforma en un conjunto de declaraciones formales de especificaciones que deben ser correctas por definición.

¿Qué es un Sistema de Programación Lógica?

El conjunto de hechos y reglas constituye lo se denomina una *descripción,* que llega a ser un programa donde combinado con un *procedimiento de inferencia* independiente de la aplicación hace posible a la computadora sacar conclusiones acerca del área de aplicación y responder preguntas aún cuando estas respuestas no estén explícitamente registradas en la descripción. Esta capacidad es la base de la tecnología de la programación lógica. La figura número 1 ilustra la configuración de un típico sistema de programación lógica. En el corazón del sistema está su procedimiento de inferencia, el cual acepta consultas del usuario, accede a los hechos y reglas en su base de conocimiento (la descripción), y saca conclusiones apropiadas. Esto es, es capaz de responder preguntas del usuario. El funcionamiento de un Sistema de Programación Lógico se basa en los conceptos de Descripción (semántica declarativa), Deducción (procesos de inferencia, deducción y resolución) y Control (eficiencia). Para una descripción detallada de éstos, véase lo propuesto por Genesereth et al. [8].

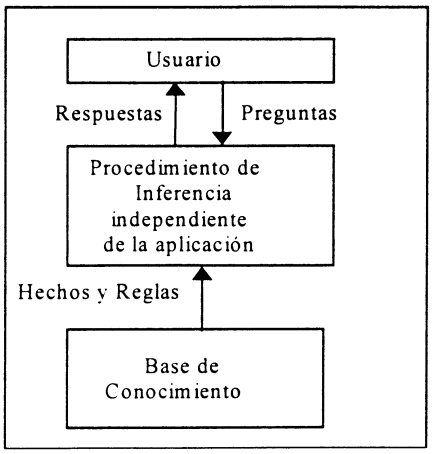


Fig. N° 1: Sistema de Programación Lógica

PROPUESTA

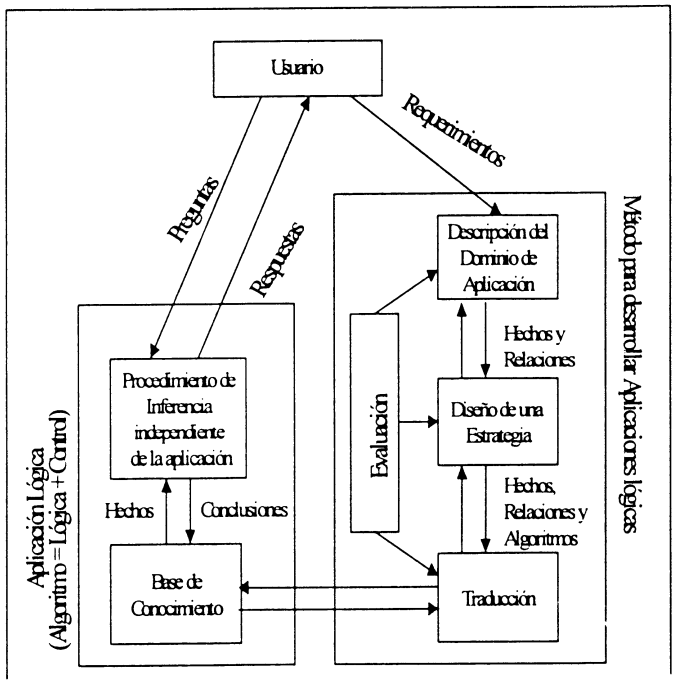
Hasta el momento se han descrito conceptos y componentes de un sistema de programación lógica. Ahora se describirá un enfoque sistemático que nos apoye en el desarrollo de aplicaciones en este paradigma. Fundamentalmente, se trata de una proposición para construir una *Descripción* apropiada, dado que los componentes de *Deducción y Control* se suponen dados. Se presenta un conjunto de fases, notación y ejemplos que muestran cómo abordar la problemática. El enfoque nos apoyará en el desarrollo de programas lógicos que permitan representar conocimiento y sean capaces de encontrar soluciones basadas en el conocimiento. El enfoque usa una notación formal para representar los conceptos necesarios y consiste en cuatro fases fundamentales: Descripción del dominio de aplicación, Diseño de una estrategia, Traducción a un lenguaje de programación que soporte el paradigma lógico, como lo es Prolog, y Evaluación del producto. Un esquema del enfoque es mostrado en la figura número 2.

Fig. N° 2: Enfoque de desarrollo propuesto

1. Descripción del Dominio de Aplicación

Definición del problema

El primer paso para desarrollar una Descripción, es establecer una *definición del problema.* Esta definición debería indicar lo que hay que hacer, y no como hay que hacerlo. Debe ser una exposición de nuestras necesidades, y no una propuesta de solución. Se debería indicar fundamentalmente qué aspectos son obligatorios, así como los opcionales. Además, señalar claramente las restricciones o limitaciones que existan. Todo esto se escribe en un documento que constituye la base para concretar la Descripción del dominio de aplicación.



A partir del documento de definición, se inicia una conceptualización de las entidades existentes o entidades necesarias y las relaciones satisfechas por estas entidades, que permitan abstraer los aspectos esenciales del dominio de aplicación. Estas entidades y sus relaciones mostrarán una visión estática del mundo.

Identificación de Entidades y sus Relaciones

Primero se identifican las entidades relevantes en el dominio de aplicación. Lo mejor es ir anotando ideas en papel antes de intentar organizarlas demasiado, aún cuando sean redundantes e incongruentes, para no perder detalles importantes. Es probable que la descripción inicial contenga defectos que deberán de ser corregidos en iteracciones posteriores. No es necesario construir uniformemente toda la descripción. Algunos aspectos del problema se podrán analizar con profundidad a lo largo de varias iteraciones, mientras que habrá otros que sigan siendo poco claros. Hay que comenzar por encontrar los candidatos a entidades que se encuentren en la Definición del problema. No hay que ser demasiado selectivo. Con frecuencia, las entidades se corresponden con nombres (sustantivo), que describen cosas concretos (por ejemplo: un circuito específico, una persona específica, este artículo) o abstractos (por ejemplo el número dos, el conjunto de todos los números enteros, el concepto justicia, libertad); ellos pueden ser primarios o compuestos (por ejemplo, un circuito que consiste de muchos componentes); ellos pueden ser hasta ficticios (por ejemplo, un unicornio). El formalismo a utilizar para describir las entidades serán los términos. El término más simple es el átomo. Un átomo puede ser utilizado para describir una entidad del dominio. Además, es posible utilizar una tuplas para describir características o atributos de las entidades. Por ejemplo:

> La entidad concreto persona "Juan" con sus atributos apellido, peso, altura, edad, ciudad de nacimiento, puede ser representado como la tupla: juan(tejada,80,1.85,35,viña-del-mar).

Los atributos deben ser identificados desde la definición del problema. Por otra parte, también debemos identificar las relaciones entre las entidades del dominio. Toda dependencia entre dos o más entidades es una relación. Una relación es una conexión física o conceptual entre entidades. Por ejemplo:

> Nuestra entidad "Juan Tejada Trabaja para la Compañía Axis". Esto puede ser expresado como: trabaja(juan, axis)
> La relación entre un País y su Capital: capital(chile, santiago)

En consecuencia, de una forma más general, se puede definir una relación como una cualidad o propiedad de un grupo de entidades con respecto a otras. Las entidades y sus relaciones pueden también ser expresadas como variables. Además, podemos expresar atributos de éstos a través de "variables", es decir, valores que no son conocidos. Por ejemplo, "todos", "algunos", "alguien", etc., pueden ser vistos como variables. La afirmación, "Todos son parecidos a su madre" puede ser representado como:

```
Parecido(Madre(Y), X).   Donde X señala una variable.
```

Hechos más complejos pueden ser escritos utilizando cláusulas con el operador ":-". La notación q :- p indica que q es verdadero si p es verdadero; en otras palabras p implica q. Estos hechos más generales son conocidos como reglas. La siguiente sentencia declara que si X es la madre de Y, entonces x es parecido a Y:

```
Parecido( X, Y ) :- Madre( X, Y )
```

Ahora sabemos como identificar y declarar entidades, sus relaciones y formas más generales como lo son las reglas. Con estos formalismos lo que se intenta al "Describir" una aplicación es representar o expresar un conjunto de estados que definen nuestro mundo en el dominio. En particular, interesará el estado inicial y el final deseado.

Ejemplo: La descripción que se hace a continuación del problema del "Granjero y el Lobo" está basada en la Descripción de Rolston[13] y Formulación de Dahal[5].

Definición del problema:

"En la mitad de un viaje, un granjero conjuntamente con su cabra, un lobo y un repollo llegan a un río que deben cruzar. Hay un bote en la rivera del río, pero solamente caven en él, el granjero y otro pasajero (la cabra, el lobo, o el repollo - presumiblemente grande y que cuenta como pasajero-). El granjero tiene que ingeniarse la secuencia de travesías del río que dé por resultado un cruce seguro, reconociendo que si se deja solo al lobo con la cabra, el lobo se comerá la cabra y a la vez si la cabra queda sola con el repollo, se comerá el repollo (Asumiremos que no se dará el caso de que el lobo se coma al granjero y luego a la cabra)".

Entidades y sus relaciones:

Hay cuatro entidades (individuos): granjero, lobo, cabra y repollo. El atributo principal de cada uno es su posición, que puede tomar valores "norte" o "sur" de la rivera del río. La primera relación es la que define el estado de nuestro dominio; es decir, el estado de cada individuo respecto de su posición: estado(X,Y,X,U). Expresa que el granjero esta en la orilla X, el lobo en la orilla Y, la cabra en la orilla Z, y el repollo en la orilla U (el orden de los argumentos es arbitrario, pero una vez elegido este debe ser respetado desde principio a fin). Claramente, los posibles valores para cada variable son "norte" o "sur" que son precisamente los valores del atributo posición de cada individuo. Una vez establecida la cláusula que define nuestro mundo, debemos declarar los estados inicial A1 y final A2 deseado:

```
A1:   estado( norte, norte, norte, norte )
A2:   estado( sur, sur, sur, sur )
```

2. Diseño de una estrategia

Se trata de diseñar una estrategia que defina un conjunto de acciones que transforman el dominio descrito por el estado inicial en otro estado, hasta llegar al estado final, preservando adecuadamente las eventuales restricciones impuestas. Para la cual, se trata en esta etapa de definir las relaciones identificadas en la etapa anterior. El problema estará resuelto encontrando el camino, utilizando las relaciones definidas, que nos llevan desde el estado inicial al estado final deseado.

Para el ejemplo: Podemos cambiar el estado de nuestros individuos colocando al granjero junto a otro viajero en una nueva posición y que sea opuesta a la actual. Así:

```
A3: estado( Y, Y, G, C) :- estado( X, X, G, C ), opuesto( X, Y )
```

Declara que si el granjero y el lobo están en la misma orilla X mientras la cabra y el repollo están en las orillas G y C, respectivamente, y si las orillas X e Y son opuestas, entonces esto hace posible alcanzar un nuevo estado en que el granjero y el lobo estarán en el

lado de la rivera opuesta a la inicial, mientras la cabra y el repollo permanecen donde ellos estaban. Esta regla representa el siguiente conjunto de transiciones.

```
A3.1: estado( Y, Y, G, C ):- estado( X, X, G, C ), opuesto( X, Y )
      (Define la acción en que el granjero cruza el río junto al lobo.
       La cabra y el repollo permanecen donde están)
A3.2: estado( Y, W, Y, C ):- estado( X, W, X, C ), opuesto( X, Y )
A3.3: estado( Y, W, G, Y ):- estado( X, W, G, X ), opuesto( X, Y )
A3.4: estado( Y, W, G, C ):- estado( X, W, G, C ), opuesto( X, Y )
```

La estrategia nos permite cambiar el estado de nuestros individuos y es recursiva La relación "Opuesto" esta dada por:

```
A4:   opuesto( norte, sur )
A5.   opuesto( sur, norte )
```

Ahora se pueden definir las condiciones sobre las cuales un estado es válido o seguro (según las restricciones dadas). Sabemos que la cabra y el lobo (o cabra y repollo) están en estado seguro, si ellos están en la misma orilla con el granjero o si ellos están en orillas opuestas. Así, podríamos definir la siguiente relación: seguro(X,Y,Z), que nos permite declarar las posiciones x e y de dos individuos en posible conflicto (cabra y el lobo o cabra y el repollo) con respecto a la posición z del granjero. Así, podemos definir las restricciones:

```
A6:   seguro( X, X, X )
A7.   seguro( X, Y, Z ) :- opuesto( X, Y )
```

Finalmente, podemos utilizar la cláusula "Seguro" para definir un estado "Aceptable". Tomemos f para la posición del granjero, w para el lobo, g para la cabra y c para el repollo, y tendremos:

```
A8: aceptable( F, W, G, C ):- seguro( G, W, F ), seguro( G, C, F )
```

Declara que si las posiciones de la cabra (g) y el lobo (w) son seguras respecto de la posición del granjero (f) y las posiciones de la cabra (g) y el repollo (c) son seguras respecto del granjero (f), entonces estamos en un estado "aceptable" para el lobo, cabra, repollo y granjero. Así, las reglas A3.i se transforman en:

```
A3.1: estado( Y,  Y, G, C ):- estado( X, X, G, C ),
      opuesto( x, y ), aceptable( y, y, g, c )
A3.2: estado( Y, W, Y, C ):- estado( X, W, X, C ),
      opuesto( x, y), aceptable( y, w, y, c )
A3.3: estado( Y, W, G, Y ):- estado( X, W, G, X ),
      opuesto( x, y ), aceptable( Y, W, G, Y )
A3.4: estado( Y, W, G, C ):- estado( X, W, G, C ),
      opuesto( x, y ), aceptable( y, w, g, c )
```

Hay un aspecto del diseño de la estrategia que no se ha considerado hasta el momento, se trata del mecanismo que permite generar el "movimiento" entre los estados posibles y válidos del mundo, partiendo de un estado inicial hasta llegar al estado final. Este punto lo analizaremos en base a un ejemplo que es tratado por Bratko [3] y Nilsson [11].

Ejemplo Dos: Definición del problema:

Hay un mono en la puerta de un cuarto. En el medio del cuarto una banana esta colgada desde el techo. El mono está hambriento y desea coger la banana, pero él no puede estirarse lo necesario a lo alto desde el piso. En la ventana del cuarto hay una caja que mono puede usar. El mono puede ejecutar las siguientes acciones: Caminar sobre el piso, encaramarse sobre la caja, empujar la caja cerca (si éste esta ya en la caja) y agarrar el plátano si la posición en la caja da directamente bajo la banana. ¿Puede el mono coger la banana?.

Entidades y sus relaciones:

El "mundo del mono" puede ser representado mediante un estado que indica la posición de los individuos (entidades). El estado inicial del mundo esta determinado por: el mono esta en la puerta, el mono esta sobre el piso, la caja esta en la ventana y el mono no tiene la banana. Así, el estado inicial es:

```
Estado(en la puerta, sobre el piso, en la ventana, no la tiene)
```

El estado inicial del mundo del mono puede ser interpretado como el estado de la posición horizontal del mono, la posición vertical, la posición de la caja, y si el mono tiene o no el plátano, en este caso no la tiene. El estado final es:

```
Estado(_,_,_,tiene)
```

Es decir, un estado en la cual el mono ha alcanzado el plátano. Hasta ahora no hemos agregado nada nuevo en el diseño de la estrategia, pero la pregunta clave es: ¿Cuáles son los siguientes movimientos que cambian el estado del mundo desde un estado a otro?. Hay cuatro tipos de movidas: caminar cerca, empujar la caja, encaramarse en la caja, agarrar el plátano. No todas las movidas son posibles en todos los estados del mundo. Por ejemplo, la movida "agarrar" es posible únicamente si el mono esta en posición sobre la caja directamente debajo del plátano (la cual esta en la mitad del cuarto) y no la ha tomado ya. Tales reglas pueden ser formalizadas de la siguiente forma:

```
Mover(Estado1,Mover,Estado2)
```

Los tres argumentos de la relación especifican una movida de la siguiente forma:

Estado1 —Mover→ Estado2

Estado1, es el estado antes de la movida, Mover es la movida ejecutada y estado2 es el estado después de la movida. La movida "agarrar", con su pre – condición necesaria sobre el estado antes de la movida, puede ser definida como:

```
Mover(Estado(medio, sobre la caja, medio, no tiene), Agarrar,
      Estado(medio, sobre la caja, medio, tiene))
```

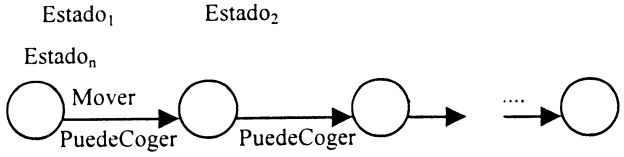
Estas relaciones muestran que después de la movida el mono tiene el plátano, y él tiene que permanecer sobre la caja en el medio del cuarto. En una forma similar podemos expresar el hecho que el mono sobre el piso puede caminar sobre el piso en una posición horizontal Pos1a una posición Pos2. El mono puede hacer esto prescindiendo de la posición de la caja y si este tiene el plátano o no. Todos estos hechos pueden ser definidos por:

```
Mover(Estado(Pos1, sobre el piso, Caja, Tiene),
Caminar(Pos1,Pos2), Estado(Pos2, sobre el piso, Caja, Tiene))
```

Los otros dos tipos de movidas, empujar y encaramarse, pueden ser especificadas de forma similar. La pregunta principal que nuestro programa tiene que responder es: ¿puede el mono en algún estado inicial coger el plátano?. Este puede formalmente expresarse como:

```
PuedeCoger(Estado)
Donde, el argumento Estado es un estado del mundo del mono.
```

La estrategia para PuedeCoger esta basada en dos observaciones: (1) Para algún estado en al cual el mono ya tiene el plátano, la relación PuedeCoger se hace verdadero; no es necesario mover en este caso. Esto corresponde a: PuedeCoger(Estado(_,_,_,tiene)). (2) En otros casos, más movidas son necesarias. El mono puede coger el plátano en algún Estado1 si hay alguna movida "Movida" desde el estado1 a algún estado Estado2, tal como el mono puede coger el plátano en el estado Estado2 (en cero o más movidas). Este principio es ilustrado en la figura siguiente:



Formulación recursiva de PuedeCoger, puede ser como: PuedeCoger(Estado1):- Mover(Estado1, Mover,Estado2), PuedeCoger(Estado2)

Esto completa nuestra estrategia la cual se muestra a continuación:

```
M1:  mover(estado(medio, sobrecaja, medio, notiene),
     agarrar, estado(medio, sobrecaja, medio, tiene)).
M2:  mover(estado(P, sobrepiso, P, H), encaramarse,
     estado(P, sobrecaja, P, H)).
M3:  mover(estado(P1, sobrepiso, P1, H), empujar(P1, P2),
     estado(P2, sobrepiso, P2, H)).
M4:  mover(estado(P1, sobrepiso, B, H), caminar(P1, P2),
     estado(P2, sobrepiso, B, H)).
M5:  puedecoger(estado(_,_,_,tiene)).
M6:  puedecoger(Estado1):- mover(Estado1, Mover,
     Estado2), puedecoger(Estado2).

? puedecoger(estado(enpuerta, sobrepiso, enventana, notiene).
```

3. Traducción

Esta etapa tiene por objetivo transformar las relaciones definidas mediante cláusulas de la etapa anterior en sentencias de un lenguaje de programación. Es decir, construir el programa lógico. Una cuestión fundamental, relacionada con una buena programación es: ¿Qué es un buen programa? Las respuestas a esta pregunta no son triviales hay muchos criterios para juzgar cuando un programa es bueno. Generalmente son aceptados criterios como los siguientes, ver Bratko [3]: Corrección, Eficiencia, Transparencia, Legibilidad, Modificabilidad, Robustez, Documentación, entre otros. Algunos elementos que pueden guiarnos hacia una buena práctica de la programación lógica, junto a la tarea de codificación, estas son: El correcto uso de la recursión, y el uso de la generalización o parametrización, ver Bratko [3].

4. Evaluación

Se trata de evaluar el programa lógico en sus aspectos de corrección (sintaxis), validez (semántica) y usabilidad (pragmática). En la corrección debemos confirmar que la solución es correcta en sus resultados y algoritmos. En la validez, debemos confirmar que los requisitos exigidos se han satisfecho plenamente. Y por último en la usabilidad, debemos confirmar que el producto final es realmente "usable" por el usuario. La corrección puede ser evaluada analizando el código fuente generado en la traducción y la ejecución algoritmica. El análisis del código fuente puede orientarse fundamentalmente ha verificar si ha sido documentado apropiadamente y si su escritura ha seguido las normas impuestas en la definición del problema. Una técnica que puede ser empleada para evaluar la corrección algoritmica es el análisis mediante refutación sintética y analítica, para detalles de esta técnica ver Kowalski [9]. La validez puede ser evaluada mediante revisiones formales. Estas consisten en chequear el documento de la definición contra los documentos generados en la Descripción, Diseño de la estrategia y Traducción. La usabilidad, fundamentalmente, debe estar orientada a evaluar los aspectos de facilidades de uso que ofrece el software. Diálogos con el usuario, orientación prestada, y que el lenguaje utilizado e interfaz sean orientados al dominio de aplicación. Estos aspectos del software son desarrollados en la etapa de Diseño de la estrategia. Sin embargo, se debe tener en cuenta que los lenguajes de programación lógica, como Prolog, no ofrecen facilidades para la construcción de interfaces, pero es posible presentar mensajes y diálogos apropiados para guiar al usuario. Otros enfoques de verificación se puede ver en Pedreschi y Ruggieri [12] y Le Charlier at al. [10].

CONCLUSION

La programación lógica con su amplio rango de aplicabilidad y específicamente en la representación de conocimiento y para implementar aplicaciones que sean capaces de encontrar soluciones a problemas en base al conocimiento, resulta una aproximación muy útil. Sin embargo, existe carencia de formalismos metodológicos para apoyar al programador lógico en el desarrollo de "Descripciones". La propuesta descrita presenta una serie de pasos, con una anotación y conceptos asociados a cada paso, que conforman un enfoque sistemático para producir aplicaciones lógicas. Se ha demostrado su aplicabilidad para desarrollar descripciones de programas lógicos que manipulan conocimiento y resuelven problemas en base a ese conocimiento.

REFERENCIAS

[1]Alonso, F.; Segovia, J. (1995): "*Entornos y Metodologías de Programación*". Paraninfo, España.

[2]Ambler, A.; Burnett, M. y Zimmerman, B. (1992): "Operational Versus Definitional: A Perspective on Programming Paradigms", *Computer*, Septiembre.

[3]Bratko, I. (1990): "*PROLOG programming for artificial intelligence*", Addison Wesley

[4]Brewka G.; Dix J. (1998): "Knowledge representation with logic programs", *Logic Programming and Knowledge Representation. Third International Workshop, LPKR'97. Selected Papers. Springer-Verlag, Berlin, Germany*; 246 pp. p.1-51

[5]Dahal, V. (1983): "Logic programming as a Representation of Knowledge", *Computer*, Octubre.

[6]Floyd, R. (1979): "The Paradigms of Programming", *Communications Of The ACM*, Agosto, Volumen 22, número 8.

[7]Gelfond M.; Gabaldon A. (1997): "From functional specifications to logic programs", *Logic Programming. Proceedings of the 1997 International Symposium. MIT Press, Cambridge, MA, USA*; 425 pp, p.355-69

[8]Genesereth, M. y Ginsberg, M. (1985): "Logic Programming", *Communivations Of The ACM*, Septiembre, Volumen 28, Número 9.

[9]Kowalski, R. (1986): "*Lógica, Programación e Inteligencia Artificial*", Díaz de Santo, España.

[10]Le Charlier B.; Leclere C.; Rossi S.; Cortesi A. (1999): "Automated verification of Prolog programs" *Journal of Logic Programming*, vol.39, no.1-3; April-Jun; p.3-42

[11]Nilsson N. (1987): "*Principios de Inteligencia Artificial*", Ediciones Díaz de Santos, España.

[12]Pedreschi D. Ruggieri S. (1999): "Verification of logic programs", *Journal of Logic Programming*, vol.39, no.1-3; April-June; p.125-76

[13]Rolston, D. (1990): "*Principios de Inteligencia Artificial y Sistemas Expertos*", McGraw-Hill, Colombia.