LCARS un Lenguaje de Especificación de Problemas en Reconocimiento de Patrones bajo el enfoque Lógico-Combinatorio

Salvador Godoy-Calderón[a], José Fco. Martínez-Trinidad[b] y Manuel Lazo Cortés[c]

[a]Centro de Investigación en Computación, CIC-IPN, Instituto Politécnico Nacional, México.
[b]Instituto Nacional de Astrofísica Optica y Electrónica INAOE, Ciencias Computacionales, Sta María Tonanzintla, Puebla C.P. 72840, México.
[c]Instituto de Cibernética Matemática y Física, Ministerio de Ciencia Tecnología y Medio Ambiente, E#309 esq. a 15 Vedado, Ciudad de La Habana. C.P. 10400, Cuba.

Abstract

En este trabajo se presenta el lenguaje LCARS para especificar problemas de Reconocimiento de Patrones bajo el enfoque lógico-combinatorio (RPLC). Se revisa el contexto en que surge la necesidad del lenguaje y la solución que éste brinda a los problemas planteados. El desarrollo de esta herramienta computacional representa un claro ejemplo de la integración de algunas técnicas de la teoría de lenguajes formales para modelar problemas en el enfoque lógico-combinatorio. Por último, se revisan un Motor de Reconocimiento de Patrones llamado " SHELL" y un ambiente de solución de problemas llamado "STUDIO" que están siendo desarrollados a partir del lenguaje LCARS.

Keywords: Logical-combinatory approach, covering, fuzzy environments.

1 Motivación

El enfoque lógico-combinatorio del reconocimiento de patrones [1-3], como varios otros enfoques [4-6], proporciona un marco teórico para el estudio y solución de problemas de clasificación, pronóstico, optimización, soporte en la toma de decisiones, etc. Cada enfoque presenta un marco teórico distinto, fundamentado en principios filosóficos diversos y, por tanto, un conjunto de axiomas diferente.

Esta incompatibilidad de teorías genera la necesidad, para cada enfoque, de procurarse las herramientas necesarias para el desarrollo de investigaciones con su propia modalidad. Procurar estas herramientas, para cualquier disciplina, exige el desarrollo de meta-conocimiento: modelos sobre la forma de desarrollar, organizar, interpretar y manipular el conocimiento propio de la disciplina. Sólo cuando se dispone de tales modelos se pueden diseñar y construir herramientas de automatización, diagnóstico y soporte a la investigación.

En particular, en el enfoque lógico-combinatorio, varios modelos de meta-conocimiento han sido desarrollados, se encuentran publicados en diversos textos y artículos de investigación y son aplicados tanto por investigadores como por estudiantes en diversas partes del mundo [1]. Sin embargo, la gran mayoría de las herramientas computacionales construidas, han sido diseñadas para resolver instancias particulares de problemas, no bajo algún modelo general. Esta limitación se ha debido fundamentalmente a la complejidad en el manejo de dominios de definición para las variables o rasgos descriptivos y la definición de funcionales de comparación.

En este artículo se presenta el desarrollo de una herramienta computacional, fundamentada en modelos generales; un lenguaje formal de especificación de problemas. Notoriamente, el desarrollo de este lenguaje requirió de la aplicación de modelos y técnicas propias de otro enfoque distinto en el reconocimiento de patrones, el enfoque sintáctico-estructural. La característica esencial de este enfoque es modelar cualitativamente los problemas, a partir de la teoría de lenguajes formales y la teoría de máquinas. Por ello, el desarrollo de este lenguaje reviste una importancia especial, no sólo por sus aplicaciones en el enfoque lógico-combinatorio, sino como un ejemplo de integración de áreas en el desarrollo de meta-conocimiento.

2 El modelo subyacente

Exponer con detalle los modelos meta-cognitivos en los que se fundamenta el diseño del lenguaje LCARS, escapa definitivamente del alcance de este trabajo. Baste, por el momento, con una revisión acerca de algunos de los elementos más relevantes que dichos modelos incluyen. Para un estudio a profundidad consúltese [7-12].

En el enfoque lógico-combinatorio de reconocimiento de patrones (RPLC), los problemas que involucran uno o más cubrimientos presentan siempre un conjunto de elementos comunes. El enfoque incluye, por supuesto, varias metodologías y técnicas especiales para el tratamiento de estos elementos comunes. El modelo que sirve de fundamento al desarrollo del lenguaje LCARS (*Logic-Combinatory Algoritmic Recognition Specification*) contempla tres asociaciones conceptuales para cada uno de los objetos del universo en estudio:

1. su evaluación o descripción en términos de un conjunto de características o rasgos descriptivos (Descripción de los objetos).

2. su pertenencia a cada una de las posibles categorías en estudio (Pertenencia a las categorías).

3. información necesaria para evaluar, ya sea de manera cuantitativa o cualitativa, la comparación entre:

   - el valor de descripción de un objeto en un rasgo particular, y el valor de otro objeto en el mismo rasgo descriptivo (Criterios de Comparación Descriptiva).
   - el conjunto de valores de descripción de un objeto y el conjunto de valores de otro objeto (Función de Analogía entre patrones).

En términos computacionales lo anterior implica que las tres necesidades fundamentales en la especificación de problemas en RPLC pueden ser expresadas de la siguiente manera:

1. Se requiere de un formato estándar que permita expresar todos los aspectos constituyentes de un problema en RPLC, incluyendo su estructura, la descripción de los objetos y las restricciones involucradas en su solución. Este formato deberá ser suficiente para especificar problemas en entornos difusos.

2. Se requiere disponer de tipos de datos para representar el dominio de definición de cada rasgo descriptivo. Además de los tipos de datos comunes (enteros, reales, etc) debe existir la posibilidad de definir nuevos tipos de datos (tipos enumerados, rangos, etc.).

3. Se requiere poder expresar los criterios de comparación y funciones de analogía de manera tal que se puedan validar de acuerdo a los tipos de datos involucrados y se puedan realizar múltiples evaluaciones de dichos funcionales.

Durante la formalización de un problema de clasificación o de selección de rasgos, es necesario especificar elementos en dos categorías: estructurales y algorítmicos. Los elementos estructurales determinan la composición del problema; el universo de objetos en estudio, los rasgos descriptivos, el conjunto de categorías, etc. Los elementos algorítmicos, en cambio, determinan el comportamiento de los componentes; el dominio de definición de los rasgos descriptivos, los diversos funcionales de comparación, las restricciones en la solución del problema, etc.

3 Estructura del lenguaje

El lenguaje LCARS fue diseñado para satisfacer todos los requerimientos de especificación de problemas expresados arriba, tanto estructurales como algorítmicos. Estructuralmente se compone de secciones llamadas bloques, posiblemente anidados, que especifican todos los elementos componentes de un problema y la

forma jerárquica en que éstos se relacionan entre sí. Algorítmicamente contiene tipos de datos que determinan el dominio de definición de diversas variables e incluye estructuras algorítmicas de control para la especificación de funcionales de comparación.

En términos generales el lenguaje LCARS presenta las siguientes características:

1. Se trata de un lenguaje de propósito específico fuertemente estructurado y tipificado.
2. Lenguaje algorítmico completo.
3. Con funciones pre-definidas adecuadas para el área de aplicación RPLC (como especificación de valores de criterios de comparación en cualquier rasgo, etc.)
4. Con una sintaxis orientada al matemático no-computólogo.

A la especificación en lenguaje LCARS de un problema le llamaremos *programa*. Así pues, un programa LCARS es en sí mismo un único bloque llamado *Proyect* que contiene dentro de sí tres sub-bloques principales: *Problem*, *Covering* y *Patterns*. Cada bloque se delimita por su nombre y un terminador con el término *End* seguido por el nombre del bloque que termina.

```
Proyect "nombre del proyecto" is
    Problem
        ...
    End Problem ;
    Covering
        ...
    End Covering ;
    Patterns
        ...
    End Patterns ;
End Project
```

El bloque *Problem* especifica la estructura del problema, las restricciones impuestas a su solución, así como las constantes y tipos de datos involucrados en el problema:

```
Problem
    ProblemType : tipo de problema ;
    ClassStyle : naturaleza de las clases ;
    ClassInteraction : relación entre las clases ;

    Restrictions
        atributo de resticción : valor ;
        ...
    End Restrictions ;

    DataTypes
```

```
            nombre del tipo : definición ;
            ...
        End DataTypes ;
        Constants
            nombre de constante : tipo de datos := valor ;
            ...
        End Constants ;
    End Problem ;
```

El bloque *Covering* especifica todos los elementos del cubrimiento inicial del problema:

```
    Covering
        Features
            nombre del rasgo : tipo de datos ;
            ...
        End Features ;
        Descriptions
            nombre objeto : [ev rasgo#1 \ pertenencia], etc... ;
            ...
        End Descriptions ;
        Classes
            nombre de clase#1, etc... ;
        End Classes ;
        Memberships
            nombre de objeto : pert. a clase#1, etc... ;
            ...
        End Memberships ;
        Criteria
            CC [ rasgo ] : tipo_de_datos is definición ;
            ...
        End Criteria ;
        Analogy
            AF [ ] : tipo_de_datos is definición ;
        End Analogy ;
    End Covering ;
```

Por último, el bloque *Patterns* especifica los objetos a clasificar:

```
    Patterns
        nombre objeto : [ev rasgo#1 \ pertenencia], etc... ;
        ...
    End Patterns ;
```

4 Tipos de Datos

Si se pretende que LCARS realmente sirva como base para la construcción de diversas soluciones en RPLC, su diseño y especificación deben ser tan dinámicos y generales como el enfoque mismo. Los tipos de datos y estructuras internas que conforman el lenguaje deben adaptarse a la mayor cantidad posible de áreas de aplicación. Evidentemente la especificación de LCARS será revisada y actualizada continuamente para adaptarla a este entorno dinámico. Para la primera versión se ha diseñado un conjunto básico de tipos de datos agrupados en dos clases: tipos de datos de sistema y tipos de datos definidos por el usuario. Los tipos de datos de sistema son aquellos pre-definidos en el lenguaje y representan la naturaleza más común de los datos:

Nombre del tipo	Significado
Boolean	Datos de naturaleza binaria
Natural	Números naturales
Integer	Números enteros
PositiveReal	Números reales positivos
Real	Números reales
Character	Caracteres
Alphanumeric	Cadenas

Por su parte, la capacidad de admitir tipos de datos definidos por el usuario, imprime la flexibilidad necesaria para la formalización de problemas sin deterioro de la legibilidad ni la facilidad de mantenimiento a los programas construidos. Para la primera versión se han especificado tres clases de tipos de datos que pueden ser definidos por el usuario:

Nombre del tipo	Significado
List	Listas de cadenas (tipos enumerados)
Subrange	Subrangos discretos de tipos ordinales
Linguistic	Variables

Para futuras versiones de LCARS se contempla la posibilidad de agregar tipos para la representación de datos binarios (imágenes) y datos vectoriales. Esto debido a que el manejo de imágenes es una de las aplicaciones más trabajadas en cualquier enfoque del reconocimiento de patrones. Otras posibles extensiones se determinarán de acuerdo a las necesidades prácticas de la comunidad de usuarios.

En lenguaje LCARS el uso de los tipos de datos se concentra en los bloques *Problem* y *Covering*. Dentro de *Problem* el bloque *DataTypes* se usa para definir los tipos de datos de usuario para el problema especificado. En el bloque *Constants* se definen las constantes usadas en el problema, y el tipo de datos de cada una. En LCARS todas las constantes son tipificadas.

En los diferentes sub-bloques dentro de *Covering* se hace uso, tanto de los tipos de datos de sistema, como de los tipos de datos definidos por el usuario,

para especificar los dominios de definición de rasgos descriptivos, criterios de comparación y función de analogía.

Obsérvese que la sintáxis usada en el bloque *Patterns*, así como en el subbloque *Descriptions* dentro de *Covering*, está diseñada para admitir la especificación de objetos en términos difusos. De igual forma, las pertenencias especificadas dentro del bloque *Memberships* son siempre datos de tipo *PositiveReal*. Estas dos características posibilitan al lenguaje LCARS para especificar problemas en entornos difusos.

5 Comparación y Analogía

La capacidad de definir tipos de datos, criterios de comparación y funciones de analogía es, sin lugar a dudas, la característica más relevante del lenguaje LCARS y la que lo distingue de otras herramientas computacionales para solucionar problemas específicos.

Como ya se mencionó, en un programa LCARS, el bloque *Covering* contiene, entre otros, los bloques *Criteria* y *Analogy*. Es en estos bloques que se definen los funcionales de comparación con la siguiente sintaxis:

Criterios de Comparación	Función de Analogía
CC [*rasgo*] : *tipo_de_datos* **is**	**AF** [ ] : *tipo_de_datos* **is**
Variables	**Variables**
definición de variables	*definición de variables*
Begin	**Begin**
instrucciones algorítmicas	*instrucciones algorítmicas*
...	...
Return (*respuesta*) ;	**Return** (*respuesta*) ;
End;	**End;**

Dentro de cada bloque delimitado por **Begin-End** se define el funcional de comparación y el valor del funcional se regresa mediante la instrucción **Return**. Para ello se dispone tanto del conjunto de operadores aritméticos y lógicos fundamentales, como de las seis estructuras algorítmicas de control. Las estructuras de control tienen la siguiente sintaxis:

Estructuras de decisión

If (*condición*) **do**	**Case**(*variable*) **equals**
bloque	*valor#1* **do** *bloque #1*
End If ;	**end**
	valor#2 **do** *bloque #2*
	end
	...
If (*condición*) **do**	**otherwise do**
bloque #1	*bloque #n*

End Case ;

end else
bloque #2
End If ;

Estructuras de Repetición

While (*condición*) **loop**
bloque
End While ;

Loop
bloque
Until (*condición*);

For *variable* **in** [*min, max , incremento*] **loop**
bloque
End For ;

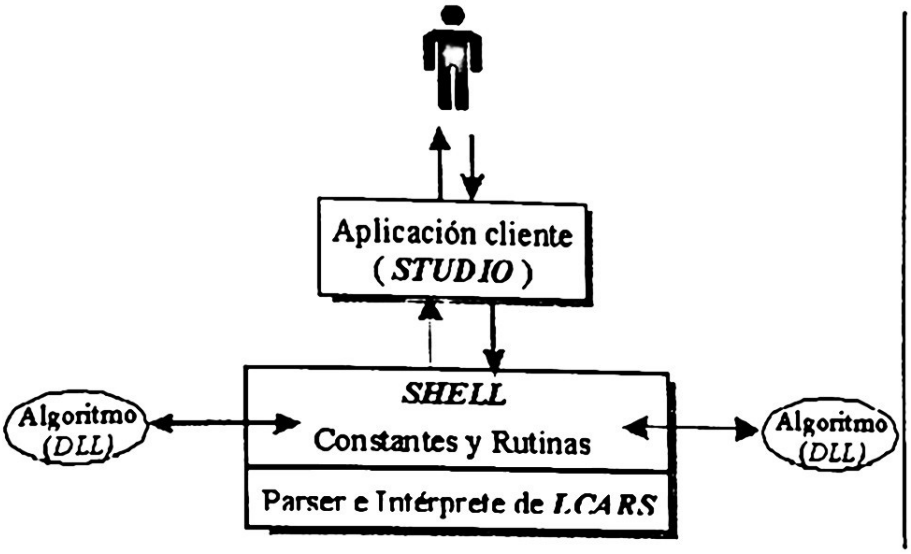
Loop
bloque
While (*condición*);

Con esto se muestra que el lenguaje LCARS efectivamente soluciona los problemas de especificación planteados en los requerimientos anteriores.

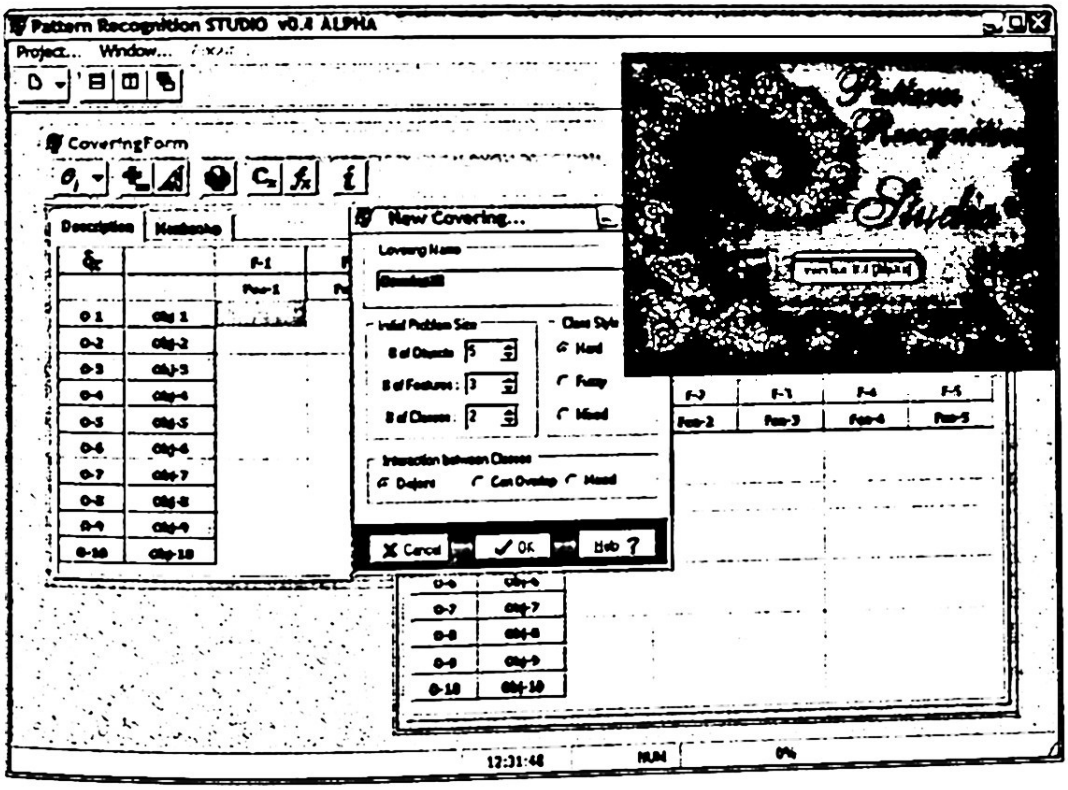
6 Construcción de un Motor de Reconocimiento de Patrones

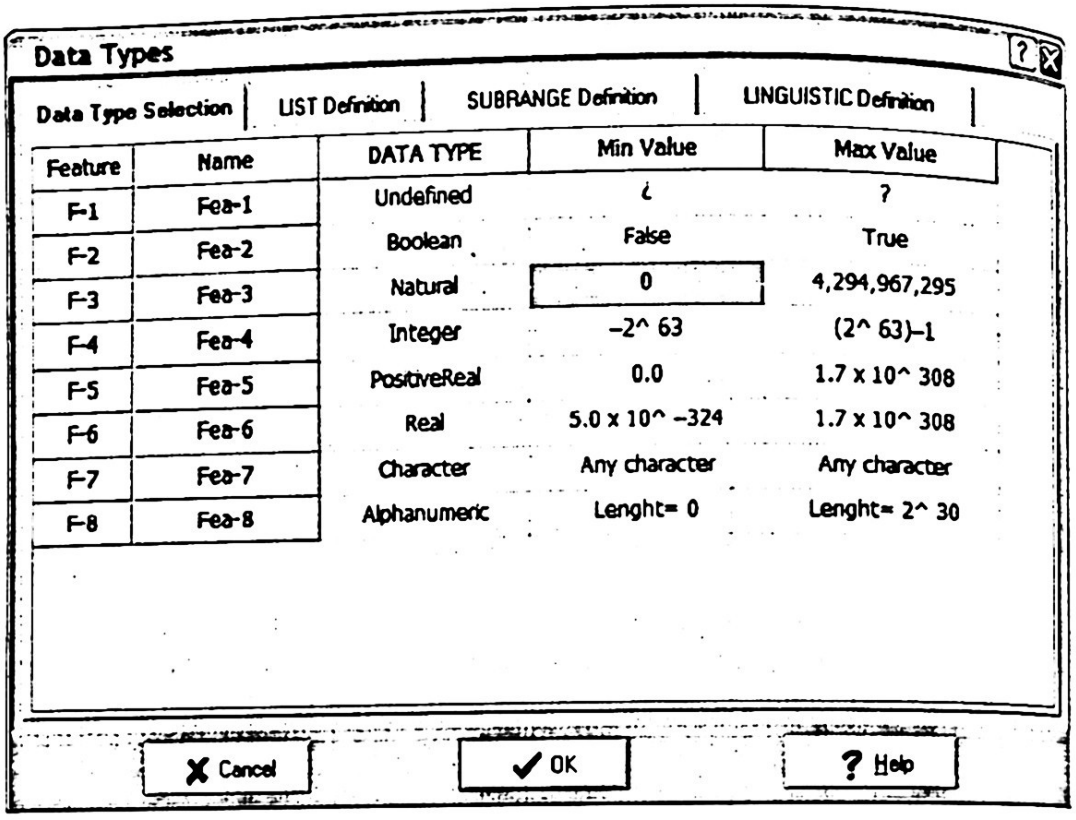
Disponer de un lenguaje de especificación de problemas no constituye una solución completa para las necesidades de investigación y docencia en RPLC. Si bien LCARS puede especificar problemas con diferentes características, es sólo eso, una herramienta de especificación. Para realmente disponer de una solución completa, es necesario agregar otra herramienta capaz de generar las estructuras de datos derivadas de la especificación (matriz de diferencias, matriz básica, etc.), así como aplicar sobre el problema especificado diversos algoritmos de clasificación, selección de rasgos, etc.

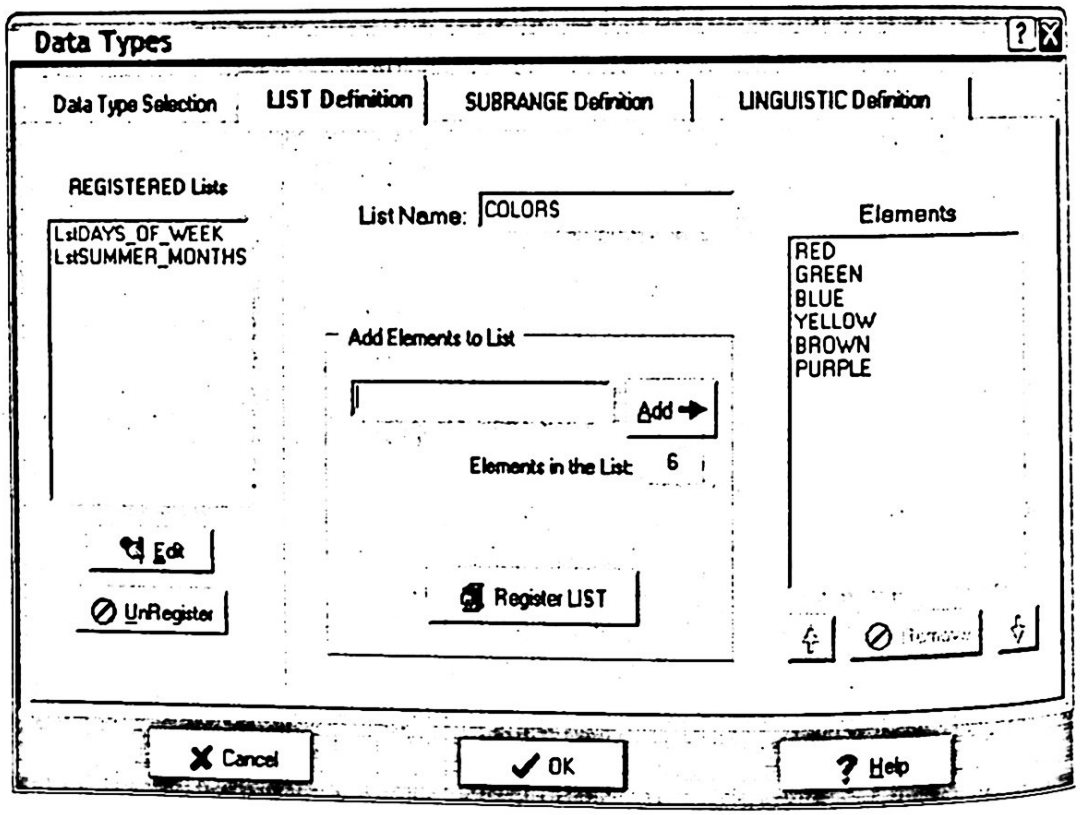
Por ello, de forma paralela a diseño de LCARS, se ha abordado el diseño y construcción de un motor de reconocimiento de patrones llamado SHELL. Este motor es una biblioteca de constantes y rutinas diseñadas para trabajar en conjunto con una especificación LCARS. De hecho, los principales elementos constituyentes del SHELL son un parser y un intérprete de LCARS. Para lograr una solución que impacte al mayor número de grupos de investigación, el SHELL se ha organizado y compilado de acuerdo con el estándar COM de *Microsoft*. Ello permitirá que cualquier investigador pueda hacer uso del SHELL desde cualquier lenguaje de programación que soporte COM. En esencia, el SHELL se ha diseñado como un componente para la construcción de aplicaciones de RPLC bajo una arquitectura cliente-servidor. En dicha arquitectura el SHELL juega el papel de servidor o motor de reconocimiento de patrones, proporciona una interfaz hacia las aplicaciones cliente (API) y tiene la capacidad de ser extendido mediante algoritmos programados en forma de bibliotecas dinámicas (DLLs). Las aplicaciones cliente pueden entonces generar soluciones de un mayor nivel de abstracción.

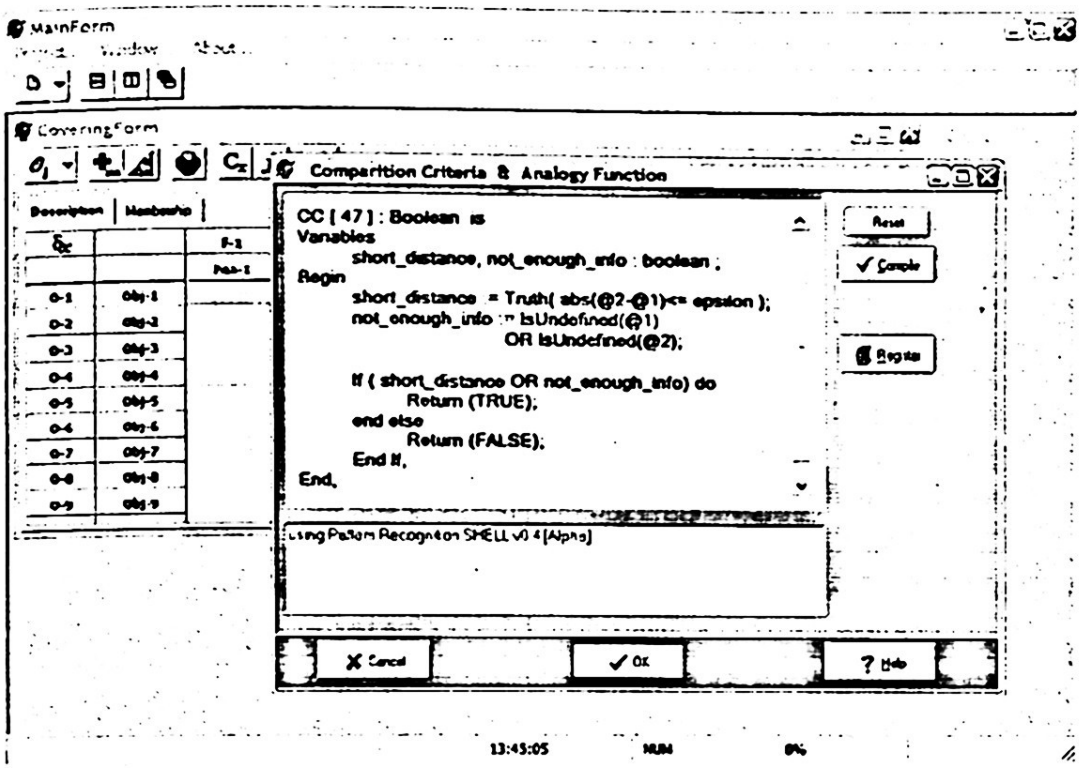


Como prueba y ejemplo de la integración del SHELL en aplicaciones de RPLC, se está construyendo también una aplicación cliente llamada Pattern Recognition STUDIO. Como su nombre lo indica, este STUDIO pretende ser un espacio para la experimentación sobre problemas y algoritmos de RPLC. Fundamentalmente se trata de una interfaz gráfica para ayudar al investigador a definir, de forma amigable, problemas en lenguaje LCARS y a procesarlos mediante las rutinas del SHELL. Algunas imágenes de este STUDIO, en su estado de desarrollo actual, se muestran a continuación.




Data Types
Data Type Selection
LIST Definition
SUBRANGE Definition
LINGUISTIC Definition
Feature
Name
F-1
Fea-1
F-2
Fea-2
F-3
Fea-3
F-4
Fea-4
F-5
Fea-5
F-6
Fea-6
F-7
Fea-7
F-8
Fea-8
DATA TYPE
Min Value
Max Value
Undefined
¿
?
Boolean
False
True
Natural
0
4,294,967,295
Integer
–2^ 63
(2^ 63)–1
PositiveReal
0.0
1.7 x 10^ 308
Real
5.0 x 10^ –324
1.7 x 10^ 308
Character
Any character
Any character
Alphanumeric
Lenght= 0
Lenght= 2^ 30
Cancel
OK
Help


Data Types
Data Type Selection
LIST Definition
SUBRANGE Definition
LINGUISTIC Definition
REGISTERED Lists
LstDAYS_OF_WEEK
LstSUMMER_MONTHS
List Name:
COLORS
Elements
RED
GREEN
BLUE
YELLOW
BROWN
PURPLE
Add Elements to List
Add
Elements in the List:
6
Edit
UnRegister
Register LIST
Cancel
OK
Help



7 Conclusiones

La actividad meta-cognitiva es un signo de madurez en cualquier disciplina. Sólo cuando se dispone de modelos meta-cognitivos en cualquier rama del conocimiento es posible evaluar el avance logrado hasta un cierto momento. ubicarlo en su justo contexto y avanzar hacia un nuevo estado con mayor nivel de abstracción que potencíe la construcción de nuevo conocimiento. Es así como evoluciona cualquier rama del saber humano y en particular, la forma en que las matemáticas han evolucionado. Para lograr ese cambio es necesario sistematizar e incluso automatizar aspectos de menor nivel de abstracción y aprovechar esa sistematización para la construcción de nuevos modelos más abstractos.

La computadora es precisamente un dispositivo para la automatización de procesos, por ello, representa una gran ayuda para la construcción de herramientas como el lenguaje LCARS y el SHELL que se pretende jueguen ese papel de impulsores del avance en RPLC. Una vez terminada la construcción de estas herramientas, su éxito dependerá de la administración tecnológica que de ellos se haga. Si y sólo si se logra construir una verdadera comunidad epistémica alrededor de ellas, será posible mantenerlas en sincronía con las necesidades tecnológicas de RPLC así como en constante actualización.

8 Referencias

[1] Ruiz-Shulcloper, J. & Mongi, A. .A. (2002), Logical Combinatorial Pattern Recognition: A Review. In *Recent Research Developments in Pattern Recognition*, ed. Pandalai, Pub. Transword Research Networks, USA., (To appear).

[2] Ruiz-Shulcloper J., Guzman-Arenas A., & Martínez-Trinidad J. F. (1999). *Logical Combinatorial Approach to Pattern Recognition: I. Feature selection and supervised classification.* Editorial Politecnica, Mexico.

[3] Martínez-Trinidad J. F.& Guzmán-Arenas A. (2001). The logical combinatorial approach to pattern recognition an overview through selected works, *Pattern Recognition* 34(4), 741-751.

[4] Fukunaga K. (1990). *Introduction to Statistical Pattern Recognition*, Academic Press, London.

[5] Duda R. O., Hart P. E., & Stork D. G. (2000). *Pattern Classification* (2nd ed), Wiley, New York , NY.

[6] Schalkoff Robert J. (1992). *Pattern Recognition: Statistical, Structural and Neural Approaches*, John Wiley & Sons, Inc.

[7] Ruiz-Shulcloper J. & Lazo-Cortés M. (1999). Mathematical Algorithms for the Supervised Classification Based on Fuzzy Partial Precedence, *Mathematical and Computer Modelling*, 29(4), 111-119.

[8] Godoy-Calderon S. (1996). "Generalization of Testor and Typical Testor concepts for fuzzy environments". M. Sc., Thesis, CINVESTAV-IPN, Mexico City, (In Spanish).

[9] Lazo-Cortes M., Ruiz-Shulcloper J. & Alba-Cabrera E. (2001). An overview of the evolution of the concept of testor, *Pattern Recognition*, 34(4), 753-762.

[10] Alba-Cabrera E. (1998). "Nuevas extensiones del concepto de testor para diferentes tipos de funciones de semejanza". Tesis Doctoral. Universidad de la Habana, Cuba.

[11] Carrasco-Ochoa J. A. (2001). "Sensibilidad en Reconocimiento Lógico Combinatorio de Patrones". Tesis Doctoral, Centro de Investigación en Computación, IPN, México, D.F.

[12] Sánchez-Díaz G. (2001). "Desarrollo de algoritmos para el agrupamiento de grandes volúmenes de datos mezclados". Tesis Doctoral, Centro de Investigación en Computación, IPN, México, D.F.