RESUMEN DE TESIS DOCTORAL

Un Paradigma Proactivo Orientado A Objetos

An object-oriented proactive paradigm

Juan Carlos Sarmiento Tovilla
Graduado en Julio 02, 2009
Centro de Investigación en Computación
Av. Juan de Dios Bátiz s/n Esq. Miguel Othón de Mendizabal C.P. 07720 México D.F.
jctovilla@cic.ipn.mx
jctovilla@gmail.com
Director: **Juan Luis Díaz de León Santiago**
Co-director: **Juan Carlos Chimal Eguía**
Centro de Investigación en Computación
Av. Juan de Dios Bátiz s/n Esq. Miguel Othón de Mendizabal C.P. 07720 México D.F.
jdiaz@cic.ipn.mx, chimal@cic.ipn.mx

Resumen. En la actualidad algunos investigadores conciben que los lenguajes como C++, Java y C# poseen una orientación interactiva. Esta forma de programación por lo regular, tiende a generar costos innecesarios en el desarrollo, en el diseño y principalmente en el manejo de los mensajes; lo que implica que el desarrollador deba tener un conocimiento extra del problema al aplicar una reingeniería de software. En este documento se presenta un enfoque basado en la computación *proactiva* e *incremental*, la cual busca que los objetos o dispositivos interactúen en beneficio del ser humano. Es por esta razón, que surge la necesidad de desarrollar y formalizar la base de *un paradigma proactivo orientado a objetos*, es decir, el paradigma propuesto da una alternativa para resolver algunos problemas que requieren ser incrementales tomando como base el paso de mensajes. Esta representación agrega reglas al paradigma orientado a objetos, lo que permite a éstos comunicarse por sentencias llamadas: *Activadores* y *Activados*.
Palabras clave: Objetos proactivos, Semántica operacional, Diseño de patrones, Objetos funcionales, Objetos Imperativos.

Abstract. At present, some researchers consider that languages like C + +, Java and C # have an interactive guide. The use of interactive programming by developers often produces unnecessary system development's costs; this involves the developer to extra knowledge when applying the software re-engineering. This paper presents an approach based on proactive computing, which looks electronic devices to interact in benefit of the human being. Due to this need, we developed and formalized the base of an object-oriented proactive-paradigm. That is, the proposed paradigm provides an alternative to solve some problems that need to be incremental, based on the passage of messages. This perspective adds rules to the object-oriented paradigm, which allows itself the objects to communicate by called methods: *Activators* and *Activated*.
Keywords: Proactive objects, Operational semantic, Patterns design, Functional objects, Imperative objects.

1 Introducción

La programación orientada a objetos ha tenido énfasis en los últimos años en la realización de los sistemas basados en cómputo. Los lenguajes orientados a objetos fueron diseñados para proporcionar una intuitiva forma de ver los datos, así como el cómputo de una manera unida. Esto permite crear una representación entre el software y el mundo de los objetos físicos [1].

De talante intuitivo se puede observar que los objetos del mundo físico interactúan entre sí. Esta interacción se realiza de diferentes maneras, ya sea por eventos, secuencias o concurrencias. En esta parte del trabajo se darán a conocer los conceptos básicos necesarios para desarrollar un paradigma de programación llamado β_{ζ}-*Cálculo*; en el que se involucra la teoría orientada a objetos. Si bien es posible decir que mucho de lo que se modela en el mundo es a través de objetos, también hay que tener en cuenta la forma en que éstos se relacionan o se ven afectados por su entorno. Este trabajo se enfocará a analizar esta interacción y cómo estos objetos pueden ser afectados. Esta forma de interacción se presentará mediante la lógica de primer orden.

El término proactivo fue acuñado por Viktor Frankl [3]. Este término se ha llevado a diferentes áreas de la sociedad, desde lo administrativo hasta el campo de la computación; donde el sentido de esta palabra toma sus peculiaridades. Hoy, la investigación en el campo de la informática se encuentra enfocada en un modelo interactivo de cómputo, esto se debe a que las personas interactúan directamente uno a uno con sus computadoras [2,6].

Con el paradigma de cómputo proactivo se pretende que las computadoras se anticipen a las necesidades del usuario y que éstas permitan tomar decisiones a nuestro favor. Esto es, mientras las personas están trabajando, las computadoras interactuarán unas con otras en busca de la solución a algún problema. Esto puede propiciar una proactividad en la actividad humana.

En el campo de la computación proactiva actualmente existen desafíos importantes que se deben resolver tales como: la conexión física de millones de nodos, modelos de cómputo, lenguajes y paradigmas. En este trabajo la palabra *proactivo* o *proactiva* tiene que ver con la noción de estar a favor de la acción, más que el significado de qué hacer con la acción misma.

Los retos que se proponen en la computación proactiva nos hacen definir un paradigma de programación orientado a objetos. Esta propuesta de paradigma deberá permitir la interacción de los objetos, lo que promueve la proactividad entre los sistemas.

Los paradigmas actuales de la programación orientada a objetos se preocupan por la clasificación jerárquica, esto nos lleva a otro planteamiento. Dentro de un modelo jerárquico existe la evolución de los objetos como lo describe Darwin. Desde un punto de vista particular, si un objeto de jerarquía superior es modificado, ¿Los objetos derivados o de jerarquías inferiores dependientes serán modificados? Dentro de la etapa de diseño de cualquier sistema esto es admisible, pero en el momento en que se trata de algo ya implantado esto tiene connotaciones colaterales. Muchos lenguajes orientados a objetos han agregado a sus clases/objetos mecanismos para indicar que algunos métodos han dejado de operar o se encuentran derogados.

Estas soluciones favorecen en mucho al trabajo de la reingeniería, aunque dentro del enfoque proactivo éstas no son favorables. Esto es un factor importante que promueve el análisis de los objetos basándose en ciertas condiciones y reglas. Estas reglas deben permitir que un objeto evolucione sin llegar a permear con los objetos de su entorno de manera directa en el diseño en su creación.

Por un momento imaginemos un sistema planetario como es el nuestro, donde existe un sol y varios planetas que giran entorno a él. Si agregamos un objeto con suficiente masa en algún instante, muchos de estos planetas se verían afectados. Esto se debe a la fuerza gravitacional y a la atracción que existe entre ellos, este efecto también se da, si retiráramos algún planeta del sistema solar descrito. Si se le pidiera a un grupo de desarrollo realizar dicho modelo físico, tendría que utilizar una serie de abstracciones comúnmente conocidas como interfaces de diseño en programación. Lo que lleva a tener que prever de alguna manera el posible comportamiento del modelo físico y conocer desde un inicio, las posibles condiciones en las que podría funcionar el sistema.

Se pretende proponer un lenguaje que nos permita modelar los sistemas físicos antes mencionados. La idea básica es que los objetos se agreguen y retiren del entorno en cualquier momento, con las reacciones que se puedan desencadenar dentro del sistema. Esto tendrá dos ventajas principales en el diseño de software: El primero es que el desarrollo del modelo puede darse de manera incremental, lo que permite dar una forma simple de evolución. La segunda, es al momento de retirar cualquier objeto en cualquier momento sin afectar de manera directa al sistema; logrando con esto, una dependencia en el diseño de los objetos.

Para ver lo explicado en el párrafo anterior, primero se lleva a cabo un análisis de los lenguajes de programación orientados a objetos, donde se deben tomar en un sentido estricto para resolver estos problemas. Un ejemplo que ilustra el problema es el de un estanque de agua con tres sensores que detecta tres niveles (n_1, n_2 y n_3), donde n_1 es el nivel bajo, n_2 es un nivel aceptable y n_3 nivel alto (peligro de desbordamiento). Se solicita realizar un sistema basado en objetos que simule dicho proceso físico, con la restricción que al momento en que el agua llegue al nivel n_3 se activa una alarma a_1.

Para mostrar cómo debe operar este paradigma se iniciará con un ejemplo. Éste se puede observar en la figura 1. Posteriormente se propone dar las bases formales para el modelado de este paradigma propuesto. Por el momento y para analizar el diagrama de la figura 1 bastará con decir que existe

una relación entre dos métodos de clases diferentes. Esta relación se encuentra basada con operaciones bajo la lógica de orden cero.

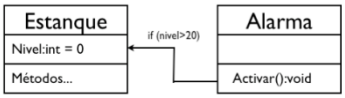

Fig. 1. Relación entre dos objetos.

En la figura 1 se tiene al objeto denominado Estanque que realizará actualizaciones en su atributo *Nivel* y que el objeto *Alarma* se activará al momento en el que el valor del *Nivel* sea mayor a *20*. Es posible mostrar el problema en un código similar al de *Java* o *C#* el cual puede quedar como se observa en la figura 2. Este código no se puede implantar de manera directa en estos compiladores, pero sirve para mostrar de una manera general lo que se pretende hacer. No obstante, sus implicaciones, características y ventajas son descritas en secciones siguientes.

```
Object Estanque [
      int Nivel = 0;
      void sensar() [
            Nivel = medir();
      ]
]
Object Alarma [
      if( Estanque.Nivel == 20)
            Activar();
      void Activar() [
            print("Alarma activada");
      ]
]
```

Fig. 2. Código abstracto para activar los objetos.

Se puede observar en la figura 2 y específicamente en el objeto llamado *Alarma*, que existe una nueva regla o instrucción denominada *if*, la cual verifica que si el *Estanque* toma un *Nivel* de *20* se ejecuta la función *Activar* del objeto *Alarma*. Este diseño permite delegar la responsabilidad de activación al objeto cliente (que es *Alarma*); mientras que el objeto *Estanque* es independiente de cualquier relación explícita en el diseño. Esto dará mayor flexibilidad y expresividad al diseñar las clases u objetos. Debido a que el diseñador establece las condiciones que activan a los objetos en estudio. Por otro lado, también se pueden realizar actualizaciones a los objetos que dependen de su entorno o contexto. Esto último promueve un sentido a la *evolución* libre de las activaciones.

1.1 Definición general de un paradigma proactivo orientado a objetos

Para realizar una definición a la propuesta de este documento, se han tomado algunos aspectos de sigma-cálculo (ς-*Cálculo*). Lo que ha originado una definición que se llamará $\beta\varsigma$-*Cálculo*. En esta última, se definirá una función *m* que se localiza en el cuerpo del objeto, así como la especificación del método de actualización. Este permite activar los objetos cercanos o dentro de su contexto. Es posible crear una condición similar a una sentencia *if* con base al ς-*Cálculo*, pero existe un motivo para separarlos, y es el llamado *activador*; además de las implicaciones en el diseño de un sistema de cálculo. Al sistema descrito se le llama de *evolución débil*, en el sentido que no se reducen los cuerpos de los métodos.

Para $\beta\varsigma$-*Cálculo* existe un cálculo basado en objetos que consiste en un conjunto mínimo de constructores sintácticos y reglas de cálculo. En esta sección se mostrará de manera informal la estructura que compone dicho cálculo para el paradigma proactivo orientado a objetos, propuesto en este trabajo.

En la tabla 1 se presenta un resumen de la noción usada para los objetos proactivos.

Tabla 1. Noción del paradigma proactivo orientado a objetos.

Expresión	***Descripción***
$\varsigma(x)b$	Método *Self* con parámetro *x* y cuerpo *b*.
$[l_i=\varsigma(x_i)b_i^{i\in 1..n}, m_j=\varsigma(x_j)if(c_j)a_j{:}b_j^{j\in n+1..m}]$	Objeto con *n* métodos de l_1 a l_n y con activaciones de m_{n+1} a m_m.
$o.l$	Invocación de un método *l* del objeto *o*.
$o.l \Leftarrow \varsigma(x)b$	Actualización del método *l* del objeto *o* con el método $\varsigma(x)b$.
$o.m \Leftarrow \varsigma(x)if(c)a{:}b$	Actualización de la activación *m* del objeto o por la condición de activación $\varsigma(x)if(c)a{:}b$.

Para más detalle, un objeto de la noción mostrada en la tabla 1 es una colección de componentes $l_i=\varsigma(x_i)b_i$. Con distintas etiquetas l_i; asociados con los métodos $\varsigma(x_i)b_i$, para $i\in 1..n$; el orden de los componentes no importa. El símbolo ς (sigma) es utilizado como un enlace (*binder*) con un parámetro (*self parameter*) de un método. También se presenta una colección de condiciones de activación $m_j=\varsigma(x_j)if(c_j)a_j:b_j$, para distintas etiquetas m_j asociados a dos expresiones *a* y *b*; el orden de estas etiquetas no importa y al igual que las etiquetas l_i el símbolo se usa como un enlazador.

Una invocación de método es escrito de la forma *o.l* donde *l* es una etiqueta del objeto *o*. La intención es ejecutar el método llamado *l* de *o* con el objeto relacionado al parámetro *Self* y devolver un objeto por la reducción.

La actualización de un método se escribe de la forma $o.l\Leftarrow\varsigma(x)b$. La semántica de la actualización es funcional: una actualización produce una copia de *o* donde el método *l* es sustituido por $\varsigma(x)b$, además de activar a todos los método *m* que se encuentren definidos en el contexto y que tengan relación con el objeto *Activador*.

Un ejemplo del uso de dicha noción se puede observar en la definición (1) con los objetos *a* y *c*.

$$a:=[\ l_1=\varsigma(x)b,\ l_2=\varsigma(x)x.l_1\Leftarrow\varsigma(x)c] \quad (1)$$

$$c:=[l_1=\varsigma(x)b,\ m_1=\varsigma(x)if(a.l_1)x.l_1:[]]$$

En el objeto *a* definido en (1) se tienen dos métodos l_1 y l_2, donde el primero hará la reducción de *b* y el segundo hará una actualización de $\varsigma(x)x.l$ por $\varsigma(x)c$. Al momento de realizar dicha actualización se activarán todas las condiciones de activación *m* que existan en el contexto y que tengan una relación con el objeto que los activa.

En el objeto *c* definido en (1) se tiene un método l_1 que reduce a *b* y una condición de activación m_1 que está en espera de ser activado. En caso de ser verdadera la expresión $a.l_1$ se ejecutará $x.l_1$. En caso contrario se ejecuta [] (objeto vacío), que en este caso expresa que no existen reducciones.

Para iniciar con la definición de la sintaxis de $\beta\varsigma$-*Cálculo*, se dará la definición de variables libres (*FV* por sus siglas en inglés) y la sustitución ($b\{x\leftarrow a\}$) para los términos llamados $\beta\varsigma$-*Término*, ver la tabla 2.

Tabla 2. Definición de variables libres

Definición
$FV(\varsigma(x)b)\equiv FV(b)-\{y\}$
$FV(x)\equiv\{x\}$
$FV([l_i=\varsigma(x_i)b_i^{\,i\in 1..n}, m_j=\varsigma(x_j)if(a_j)b_j:c_j^{\,j\in n+1..m}])\equiv$ $(\cup^{i\in 1..n} FV(\varsigma(x_i)b_i))\cup(\cup^{j\in n+1..m} FV(\varsigma(x_j)if(a_j)b_j:c_j))$
$FV(a.l)\equiv FV(a)$
$FV(a.l\Leftarrow\varsigma(y)b)\equiv FV(a)\cup FV(\varsigma(y)b)$
$FV(\varsigma(y)if(a)b:c)\equiv FV(a)\cup FV(b)\cup FV(c)-\{y\}$
$FV(a.m\Leftarrow\varsigma(x)if(a)b:c)\equiv FV(a)\cup FV(\varsigma(x)if(a):b:c)$

El propósito de la siguiente teoría de ecuaciones es capturar la noción de igualdad. Esto se usará para definir el momento en que dos objetos son de la misma forma. Se agregarán las reglas: simétrica, transitiva y de congruencia (esta última se utiliza para sustituir iguales por iguales), ver tabla 3.

Tabla 3. Teoría de ecuaciones

Expresión	***Descripción***
$b\Leftrightarrow a\perp a\Leftrightarrow b$	Simétrica
$b\Leftrightarrow a\ b\Leftrightarrow c\perp a\Leftrightarrow c$	Transitiva
$\perp x\Leftrightarrow x$	Congruencia
$b_k\Leftrightarrow b_i\ a_k\Leftrightarrow a_i\ c_k\Leftrightarrow c_i\ d_k\Leftrightarrow d_i$ $\forall i\in 1..n\ \ \forall j\in n+1..r\perp x$ $=[l_i=\varsigma(x_i)b_i^{\,i\in 1..n}$ $,m_j=\varsigma(x_j)if(c_j)a_j:b_j^{\,j\in n+1..m}]$ $\Leftrightarrow x$	Objeto
$a\Leftrightarrow a'\perp a.l\Leftrightarrow a'.l$	Selección
$a\Leftrightarrow a'\ b\Leftrightarrow b'\perp a.l$ $\Leftarrow\varsigma(x_i)b_k\Leftrightarrow a'.l\Leftarrow\varsigma(x)b'$	Actualización

2 Definición del paradigma propuesto

Los lenguajes imperativos son una abstracción subyacente a la máquina de Von Neumann [4], en el sentido que ellos conservan las partes esenciales básicas sin detalles superfluos. Una vista jerárquica es que los lenguajes de bajo nivel proveen un limitado nivel de abstracción, mientras un lenguaje de alto nivel puede ser visto como una máquina virtual. En esta última de manera general, se pueden encontrar algunas manipulaciones sobre la memoria dada por algunas entradas y salidas.

Estas por lo regular son expresadas de manera independiente al hardware en que se pretenda implantar. Los lenguajes orientados a objetos son naturalmente imperativos [5], con métodos que realizan alguna operación dentro o fuera de los objetos.

En esta sección se analizará la definición imperativa de $\beta\varsigma$-*Cálculo* mediante la formalización basada en la semántica operacional. Se desarrollará un pequeño, pero expresivo lenguaje imperativo $\beta\varsigma$-*Cálculo*; que es una variante del lenguaje funcional presentado en la tabla 1. Este lenguaje imperativo será el núcleo para el intérprete desarrollado y llamado *Pro-Objects*, con lo que es posible experimentar con aspectos importantes de la programación orientada a objetos, el cual incluye definiciones, reducciones y vistas. La sintaxis esencial de *Pro-Objects* se encuentra en la tabla 1 más algunas expresiones mostradas en la tabla 4.

Tabla 4. Extensión para la sintaxis de Pro-Objects

Expresión	***Descripción***
a, *b* y *c*	Variables
clone(*a*)	Clona el objeto a.

Pro-Objects realiza la actualización de un componente se realiza mediante $o.l \Leftarrow \varsigma(x)b$. Esto buscará el método *l* del objeto *o*, para luego reemplazarlo por el lado derecho, que es $\varsigma(x)b$. Una vez realizada la actualización, la máquina abstracta procede a buscar todos los mecanismos de activación *m* los cuales verifican la operación $o.m \Leftarrow \varsigma(x)if(a)b:c$. Este último indica que la etiqueta *m* del objeto o será actualizado por $\varsigma(x)if(a)b:c$. Como información, una diferencia con la actualización que presenta ς-*Cálculo*, es que esta última activa a los objetos que se encuentran dentro del mismo contexto, además de hacer el reemplazo de la reducción de la parte izquierda por la parte derecha de dicha operación.

El método *clone*(*a*) es la función que se encarga de realizar una copia en profundidad de un objeto. Esta copia se podría realizar de tres formas: La primera es llamada copia superficial; ésta no copia referencias o instancias internas al objeto. La segunda es denominada copia en profundidad; ésta hace una copia de todas las referencias o instancias internas al objeto. El tercero y último es conocido como clonación mixta en métodos, en la que intervienen las dos primeras formas; clonación superficial y en profundidad.

Para complementar la sintaxis se presentan las reglas semánticas de $\beta\varsigma$-*Cálculo*.

3 Semántica operacional de βς-Cálculo

La semántica operacional en (2) es expresada en función de una relación. Esta relación se encuentra dada por un sistema de almacenamiento σ (*Store*) y una pila *S* (*Stack*) y con un término *b* que se reduce a *v*, este último se coloca en σ'.

$$\sigma \cdot S \text{ a } b \Rightarrow v \cdot \sigma' \tag{2}$$

La intención de realizar ésto, es que se inicie con el almacenamiento σ (*heap*) y la pila *S*, el término *a* reduce a un resultado *v*, cediendo una actualización al almacenamiento σ' y dejando la pila *S* sin cambios. Las siguientes entidades implicadas en la semántica pertenecen a las clases definidas en la tabla 5:

Tabla 5. Definición de almacenamiento y pila

Expresión	***Descripción***
$\iota \in Nat$	Localización de almacenamiento.
$v ::= [l_i = \iota_i , m_j = \iota_j]^{\,i,j \in ..n}$	Resultado (l_i y m_j) distintos.
$S ::= (x_i \rightarrow v_i)^{\,i \in ..n}$	Pila con x_i distintas.
$\sigma ::= (\iota_i \rightarrow \langle \varsigma(x_i) b_i^{\,i \in 1..n}, S \rangle,$ $\iota_j \rightarrow \langle \varsigma(x_j) if(c_j) a_j : b_j^{\,j \in 1..m} \rangle$	Almacenamiento (store) (l_i y m_j) distintos.
σ a $\Diamond$	Juicio para almacenamiento
$\sigma \cdot S$ a $\Diamond$	Juicio para la pila
$\sigma \cdot S$ a $b \Rightarrow v \cdot \sigma'$	Juicio para la reducción de términos.

Un resultado *v* representa un objeto el cual muestra una colección de nombres de métodos, junto con la localización correspondiente en la que son colocados los métodos cerrados. También se puede apreciar una colección de nombres de *condiciones de activación*, que al igual que los métodos, presentan una ubicación donde están almacenadas las condiciones de activación cerradas.

Un método cerrado se encuentra construido por $\varsigma(x)b$ y una pila S_i, tal que $FV(\varsigma(x_i)b \subseteq dom(S_i))$. Finalmente, esta representación se encuentra asociada a la localidad de memoria.

Asimismo, una condición de activación cerrada se encuentra definida por $\varsigma(x_i)if(c_i)a_i:b_j$ y una pila S_i, tal que $FV(\varsigma(x_i)if(c_i)a_i:b_i) \subseteq dom(S_i)$, y a su vez asociada a una localización de memoria.

A continuación se describirán algunos aspectos para el almacenamiento y sustitución realizados por la máquina abstracta, con base a las siguientes expresiones:

Representación de la relación de almacenamiento entre ι_i y su término para $i \in 1..n$, ver tabla 6:

Tabla 6. Relaciones de almacenamiento

Término
$\iota_i \rightarrow \langle \varsigma(x)b,S\rangle_i$
$\iota_i \rightarrow \langle \varsigma(x)if(c)a:b,S\rangle_i$
$\iota_i \rightarrow \langle true \mid false,S\rangle_i$

Representación de la relación de colocar el resultado del término cerrado en la localidad ι_i de σ para $i \in 1..n$, ver tabla 7:

Tabla 7. Relaciones de almacenamiento

Término
$\sigma . \iota_i \rightarrow \langle \varsigma(x)b,S\rangle$
$\sigma . \iota_i \rightarrow \langle \varsigma(x)if(c)a:b,S\rangle$
$\sigma . \iota_i \rightarrow \langle true \mid false,S\rangle$

Definiciones iniciales para los esquemas de reducción de $\beta\varsigma$-*Cálculo*, ver tabla 8:

Tabla 8. Definiciones básicas

Expresión	***Descripción***
fun: $\langle \varsigma(x)b,S\rangle$	Objeto
act: $\langle \varsigma(x)if(c)a:b,S\rangle$	Activador
bol : *true* | *false*	Constantes
obj: $[l_i=\varsigma(x_i)b_i^{j\in 1..n}, m_i=\varsigma(x_i)if(c_i)a_i:b_j^{j\in n+1..m}]$	Objeto

Estructuras básicas para la reducción de *Store*, ver tabla 9:

Tabla 9. Definiciones básicas

Expresión
Store $\varnothing$: $\perp \varnothing \rightarrow \Diamond$
Store ι: $\sigma . S \rightarrow \iota, \iota' \notin dom(\sigma) \perp \sigma, (\iota \rightarrow$ fun, $\iota' \rightarrow$act) $\Rightarrow \Diamond$

Reducciones básicas para fun-$\beta\varsigma$ e imp-$\beta\varsigma$, ver tabla 10:

Tabla 10. Definiciones básicas $\beta\varsigma$

Expresión
Red x: σ. $(S',x \rightarrow v,S'') \downarrow \Diamond \perp \sigma . (S',x \rightarrow v,S'') \downarrow x \rightarrow v . \sigma$
Red constantes: ι: $\sigma . S \rightarrow \Diamond \perp \sigma . S \downarrow$ bol$\rightarrow$bol $. \sigma$
Red objetos: $\sigma . S \downarrow \Diamond \iota, \iota' \notin dom(\sigma) \perp \sigma . S \downarrow$ obj $\Rightarrow v . (\sigma, \iota \rightarrow$ fun, $\iota \rightarrow$ act)
Red selección: $\sigma . S \downarrow a . v . \sigma' \sigma'(l_k)$=fun $x_k \in dom$(S') $\sigma'(S', x_k \rightarrow v) \downarrow b_k \Rightarrow v . \sigma'' \perp \sigma . S \downarrow a.l_j \Rightarrow v . \sigma''$
Red clonación: $\sigma . S \downarrow \sigma \Rightarrow v\ \iota_i, \iota_j \in dom(\sigma')\ \iota'_i, \iota'_j \in dom(\sigma')\ \forall i,j \in 1..n \perp \sigma . S \downarrow$ clone(a) $\Rightarrow$ obj $. (\sigma', \iota'_i \rightarrow \sigma'(l_i), \iota'_j \rightarrow \sigma'(l_j))$
Red *let*: $\sigma . S \downarrow \sigma . v . \sigma' \sigma' . (S, x \rightarrow v') \downarrow b \Rightarrow v''. \sigma'' \perp \sigma . S \downarrow$ *let* $x = a$ *in* $b \Rightarrow v'' . \sigma''$

Semántica operacional para las condiciones de activación fun-$\beta\varsigma$, ver tabla 11.

Tabla 11. Definiciones activación $\beta\varsigma$

Expresión
Red actualización: $\sigma . S \downarrow a \rightarrow v' . \sigma' \iota_k \notin dom(\sigma')$ $\sigma\beta(l_k)$=act $x_r \in dom(S')\ \forall$r $\in$ 1..n
Red *true*: $\sigma . S' \downarrow a \Rightarrow true . \sigma' \sigma' . S \downarrow$ a $\Rightarrow v . \sigma'' \perp \sigma\beta . S \downarrow$ act $\Rightarrow v' . \sigma''$
Red *false*: $\sigma . S' \downarrow a \Rightarrow false . \sigma' \sigma' . S \downarrow$ b $\Rightarrow v . \sigma'' \perp \sigma\beta . S \downarrow$ act $\Rightarrow v' . \sigma''$
Red *act*: $\sigma . S \downarrow a \Rightarrow v . \sigma' l'_k = \notin dom(\sigma')\ k \in 1..n \perp \sigma . S \downarrow a.m_k \Leftarrow$ act $\Rightarrow v . (\sigma', l'_k \rightarrow$ act)

3.1 Ejemplos de reducciones

Para el siguiente ejemplo se empleará la mayoría de las reducciones definidas en la sección anterior y se establecerá cada regla de derivación. Primero se presenta la descripción de cuatro objetos que se encuentran definidos en (3) que pertenecen al mismo contexto (*A*, *B*, *C* y *D*).

$$A=[l_1=\varsigma(x)false, l_2=\varsigma(x)true]$$

$$B=[\ l_1=\varsigma(x)if(A.l_1)A.l_2 \Leftarrow \varsigma(x)true : A.l_2 \Leftarrow \varsigma(x)false,\ l_2 = clone(A)] \quad (3)$$

$$C=[\ l_1=\varsigma(x)B.l_1 \Leftarrow \varsigma(x)if(A.l_2)A.l_1 \Leftarrow \varsigma(x)true : A.l_2 \Leftarrow \varsigma(x)false$$

$$D=[l_1=\varsigma(x)\ A.l_1 \Leftarrow \varsigma(x)true]$$

Se define un esquema de reducción en (4):

$$let\ A=[l_1=\varsigma(x)false, l_2=\varsigma(x)true]\ in$$

$$B=[\ l_1=\varsigma(x)if(A.l_1)A.l_2 \Leftarrow \varsigma(x)true : A.l_2 \Leftarrow \varsigma(x)false,\ l_2 = clone(A)]\ in \quad (4)$$

$$D=[l_1=\varsigma(x)\ A.l_1 \Leftarrow \varsigma(x)true]\ in\ D.l_1$$

Reducción por medio de selección del objeto *A* en (5):

$$\varnothing.\varnothing \downarrow [l_1=\varsigma(x)false, l_2=\varsigma(x)true] \Rightarrow [l_1 = \iota_1,\ l_2 = \iota_2]\ .\ (\iota_1 \rightarrow \langle \varsigma(x)false, \varnothing\rangle,\ \iota_2 \rightarrow \langle \varsigma(x)true, \varnothing\rangle) \quad (5)$$

Reducción de *true* en (6):

$$(l_1 \rightarrow \langle \varsigma(x)false, \varnothing\rangle,\ l_2 \rightarrow \langle \varsigma(x)true, \varnothing\rangle)\ .\ (x \rightarrow [\ l_2 = \iota_2])\ true \Rightarrow true\ .\ (\iota_1 \rightarrow \langle \varsigma(x)false, \varnothing\rangle,\ l_2 \rightarrow \langle \varsigma(x)true, \varnothing\rangle) \quad (6)$$

Un ejemplo basado en el lenguaje *Pro-Object* es el de sincronizar una serie de semáforos, ésto nos permitirá tener una visión general del funcionamiento de los objetos proactivos.

S_1=[color=rojo, m=$\varsigma(x)if(S_2$.color == rojo) *x*.color ⇐ rojo : *x*.color ⇐ verde]

S_2=[color=rojo, m=$\varsigma(x)if(S_1$.color == rojo) *x*.color ⇐ rojo : *x*.color ⇐ verde]

S_3=[color=verde, m=$\varsigma(x)if(S_1$.color == rojo) *x*.color ⇐ verde : *x*.color ⇐ rojo]

Se puede observar en el código anterior que si se aplica una reducción de la forma: *Red* S_1*.color* ⇐ *verde*. El semáforo tres S_3 quedará definido como se muestra en a continuación.

S_3=[color=rojo, m=$\varsigma(x)if(S_1$.color == rojo) *x*.color ⇐ verde : *x*.color ⇐ rojo]

Se observa en S_3 que el *color* de *verde* cambió a *rojo* y la reducción de: S_1*.color* debió quedar en *verde*.

4 Conclusiones

En este documento se buscó una breve descripción de un paradigma proactivo orientado a objetos y fundamentar las bases de los objetos bajo el concepto de activaciones.

Dentro de la búsqueda para resolver problemas de cómputo en el ámbito de lo proactivo, se formalizó y desarrolló una especificación orientada a objetos, la cual permite dar una solución a los problemas de cómputo de esta naturaleza de manera incremental. Asimismo, se describió una solución incremental para el desarrollo de software que admite a los objetos interactuar entre si.

En el desarrollo del lenguaje e intérprete, se realizaron algunos estudios e implicaciones en el área de la computación. Aunque el cómputo proactivo es un área relativamente nueva, es imperante mostrar ciertas preeminencias que hacen de este paradigma proactivo orientado a objetos una forma simple de resolver algunos problemas pertenecientes a este campo.

5 Trabajos futuros

Durante el desarrollo del presente trabajo se encontraron varios aspectos que podrían ser desarrollados como trabajos futuros. A continuación se enuncian algunos de ellos:

- Adaptar el paradigma propuesto a algún lenguaje orientado a objetos. Se pretenderá llevar esta propuesta a un lenguaje ya existente con la creación de un compilador basado en una nueva especificación. Esto se hará en función del problema que se requiera resolver.

- La formalización del paradigma proactivo orientado a objetos mediante el π-*Cálculo*. Lo que se buscará es establecer este paradigma para manejar la concurrencia.
- Promover un estudio dentro de $\beta\varsigma$-*Cálculo* para la utilización de ciertos operadores de visibilidad, que eviten conflictos con los auto-llamados o la recursividad no controlada.
- Impulsar un estudio con el uso de la herencia y analizar la evolución de los objetos bajo estos mecanismos de clasificación.
- Efectuar un estudio sobre la manera eficiente de implantar este paradigma bajo el concepto de Máquinas de Turing comparado con algún otro modelo de cómputo.
- Analizar el paradigma con los conceptos de Clases y Prototipos. Lo que se buscará es indicar las ventajas de una y otra implementación, así como el uso de métodos propios y delegados en dicho análisis.
- Desarrollar un sistema de transacciones mediante este paradigma. El objetivo general es el de resolver la restauración de valores si llegase a suceder alguna acción posterior.
- Implantar algunos problemas pertenecientes al campo de la Inteligencia Artificial con este paradigma. Por dar algunos ejemplos: redes neuronales, algoritmos genéticos, agentes, entre otros.
- Buscar otros mecanismos de activación en los objetos. Este mecanismo deberá ayudar a resolver otros problemas de diseño e implantación en el área de computación.

Referencias

1. **Abadi, M. & Cardelli, L. (1996).** *A Theory of Objects.* New York: Springer-Verlag.
2. **Clark, D. D., & Tennenhouse, D. L. (1990).** *Architectural considerations for a new generation of protocols. ACM SIGCOMM Computer communications Review, 20 (4), 200-208.*
3. **Frankl, Viktor E. (1997).** *Man's Search for Meaning. Boston: Beacon Press.*
4. **Fernandez, M. (2004).** *Programming Languages and Operational Semantics: An Introduction. Chichester: King's College Publications.*
5. **Ranta, A. (1994).** *Type theory and the informal language of mathematics.* Types for Proofs and Programs, *Lecture Notes in Computer Science, 806, 352,365.*
6. **Tennenhouse, D. L. (2000).** *Proactive computing. Communications of the ACM, 43(5), 43-50.*
7. **Sarmiento, J. C. (2009).** *Un paradigma proactivo orientado a objetos, Tesis de doctorado, Instituto Politécnico Nacional, Centro de Investigación en Computación, México, D.F.*
8. **Sarmiento, J. C. & Horta, J. M. (2007).** *Un diseño alternativo para clonar objetos en Java 1.6, Congreso internacional de sistemas computacionales, Tuxtla Gutierrez, México,*

Juan Carlos Sarmiento Tovilla

Realizó sus *estudios de Ingeniería en Sistemas Computacionales en el Instituto Tecnológico de Tuxtla Gutiérrez Chiapas. Obtuvo el grado de Maestro en Ciencias de la Computación y el de Doctor en Ciencias de la Computación en el Centro de Investigación en Computación del Instituto Politécnico Nacional. Sus áreas de interés son Teoría orientada a objetos*, *seguridad en informática e informática educativa. www.jctovilla.org.*

Juan Luis Díaz de León Santiago

Obtuvo el grado de Doctor en Ciencias en Morfología Matemática en el año de 1996 en el CINVESTAV, actualmente es Profesor-Investigador de tiempo completo en el Centro de Investigación en Computación del Instituto Politécnico Nacional.

Juan Carlos Chimal Eguía

Obtuvo el grado de Doctor en Ciencias con Especialidad en Física por la Escuela Superior de Físico Matemáticas del Instituto Politécnico Nacional, actualmente es Profesor-Investigador de tiempo completo en el Centro de Investigación en Computación del Instituto Politécnico Nacional.