CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL INSTITUTO POLITÉCNICO NACIONAL

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

SECCIÓN DE COMPUTACIÓN

OBOCCAM : OCCAM BASADO EN OBJETOS

Tesis que presenta el Lic. Raúl Hernández Cruz para obtener el grado de MAESTRO EN CIENCIAS en la especialidad de INGENIERÍA ELÉCTRICA.

Trabajo dirigido por el M. En C. José Oscar Olmedo Aguirre.

CINVESTAV/CONACYT México D.F Junio de 1997

XM

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

SECCIÓN DE COMPUTACIÓN

OBOCCAM: OCCAM BASADO EN OBJETOS

Reseña :

OBOCCAM se desarrollo como una alternativa para la explotación de la programación concurrente mediante técnicas de programación orientada a objetos.

OBOCCAM es un lenguaje de programación procedural, en el cual se definen construcciones de programación concurrente y orientadas a objetos, estas construcciones son utilizadas en el desarrollo de aplicaciones que explotan el paralelismo en entornos constituidos por TRANSPUTER.

El subconjunto de construcciones concurrentes empleadas pertenecen al lenguaje OCCAM, así OBOCCAM aumenta este conjunto con otras cuya filosofía esta basada en la programación orientada a objetos, de esta manera se conjugan la programación concurrente procedural (OCCAM basado en mensajes) y la orientada a objetos (OBOCCAM) .

Dentro de las facilidades de OBCCAM se encuentran la declaración de objetos, el polimorfismo, la construcción y destrucción de objetos, el llamado a objetos remotos, entre otros.

Las principales ventajas obtenidas con OBOCCAM son aquellas derivadas de las técnicas de la orientación a objetos como lo son el ocultamiento de la información, la reusabilidad del código, el desarrollo de componentes distribuidos, así como mejores abstracciones del mundo real.

Con respecto al entorno sobre el cual opera OBOCCAM, se encuentra el procesador TRANSPUTER (procesador de 32 bits) cuya principal característica es su capacidad de interconexión con otros procesadores similares, lo que nos permite la construcción de redes de procesadores con la capacidad de ejecutar sistemas paralelos en arquitecturas de bajo costo con topologías diversas (anillo, red, toroidal, etc.).

OBOCCAM y OCCAM están íntimamente ligados al procesador TRANSPUTER sin embargo, los resultados obtenidos son validos a otros tipos de arquitecturas, tales como redes de procesadores comerciales basados en otros tipos de procesador (Pentium, Alfa etc.)

OBOCCAM :
OCCAM BASADO EN OBJETOS

POR RAÚL HERNÁNDEZ CRUZ*

TESIS DIRIGIDA POR:
M. en C. JOSÉ OSCAR OLMEDO AGUIRRE

CENTRO DE INVESTIGACIONES Y DE ESTUDIOS AVANZADOS DEL INSTITUTO POLITÉCNICO NACIONAL

RESUMEN

En el presente trabajo, se tocan temas de programación concurrente, programación orientada a objetos y arquitecturas paralelas, todo el material presentado está enfocado hacia el objetivo primordial del trabajo, el cual consiste en extender el lenguaje OCCAM[12] para que soporte los principales conceptos de la programación orientada a objetos.

OCCAM es un lenguaje concurrente que nos ayuda a programar el procesador TRANSPUTER, el cual es un procesador de 32 bits con características tales, que nos permiten interconectarlo con otros procesadores para formar redes y así explotar el paralelismo.

Aquí se describen las características de la extensión como lo son la declaración de objetos, sus métodos, su sintaxis, la forma en que se implemento y algunos ejemplos que ilustran las características de OBOCCAM.

Palabras Claves :
Programación concurrente, lenguajes de programación, procesos, sincronización, comunicación entre procesos, procesamiento en paralelo, compiladores, OCCAM, TRANSPUTER, tipos de datos, tipos de datos abstractos, programación orientada a objetos, objetos, polimorfimo, constructores destructores, sobrecarga de operadores.

Abstract

In this work, the issues about concurrent programming ,object oriented and parallel architectures are described, every part of the material showed is focused to the main objectives, where it is defined as the extension of the language OCCAM to support the concept of object oriented programming.

OCCAM is a concurrent language helps us program the TRANSPUTER processor, which is a processor of 32 bits with characteristics that permit us to connect several processors within and to make networks of processors and therefore exploit the parallel processing.

Here we describe the characteristics of the extension, the syntax and the way it was implemented, finally we will give some examples of its properties

Key words :
Concurrent programming, programming languages, process, synchronization primitives, communication between process, parallel processing, compilers, OCCAM, TRANSPUTER, data types, abstract data types, object oriented programming, objects, polymorphism, constructors, destructors, operator Overload.

Con cariño :

A mis Padres y Hermanos

por su ayuda. . .

Agradecimientos :

Agradezco enormemente a mi asesor Oscar Olmedo por dirigir y ayudarme a terminar este trabajo, pienso que si no hubiese sido por él nunca se habría terminado.

Cabe mencionar a todas aquellas personas que me apoyaron y me animaron a seguir adelante, sobre todo, a mis amigos y compañeros del Centro de investigación, pero resultaría difícil enumerar a todos y para evitar que se me olvide alguno prefiero darles las gracias en forma general.

.

Agradezco igualmente a todos los Profesores del centro de investigación que se esfuerzan por darnos lo mejor de sus conocimientos.

Finalmente agradezco al CONSEJO NACIONAL DE CIENCIA Y TECNOLOGIA por haberme apoyado y otorgado una beca para la realización de esta Maestría.

Raúl Hernández Cruz.

OBOCCAM: OCCAM BASADO EN OBJETOS

INDICE GENERAL

INDICE DE FIGURAS

Introducción :

El presente trabajo tuvo origen en los conceptos, bibliografía, experiencias e ideas que fueron interviniendo durante el diseño e implementación del lenguaje de programación OBOCCAM.

OBOCCAM es la continuación del proyecto de investigación de O. Olmedo[8], en donde se abordan conceptos importantes de la integración natural entre la programación concurrente y la orientada a objetos.

En uno de sus trabajos, se realiza la extensión del conjunto de sentencias al lenguaje C++, cuya principal característica es la facilidad que brinda para la programación orientado a objetos, este lenguaje no tiene características de programación concurrente, con lo cual, por medio de la extensión, se le añadieron las primitivas básicas para el control de la concurrencia, con esto se lograba tener un lenguaje dentro del paradigma de la programación concurrente orientada a objetos.

A diferencia de la extensión de C++, nosotros partimos del lenguaje de programación llamado OCCAM, el cual nació con la programación concurrente, en este trabajo se explica sobre el desarrollo de OBOCCAM, el cual cubre las características de su extensión en la manipulación de objetos y de la comunicación entre ellos.

También se presenta, el tratamiento del concepto y la teoría sobre OBOCCAM y se hace énfasis en el desarrollo del esquema de comunicación entre objetos en la solicitud de servicios.

Es importante saber por que se le ha nombrado OCCAM BASADO en OBJETOS en lugar de ORIENTADO a OBJETOS esto se fundamenta en el hecho de que no soporta todos los conceptos de la programación orientada a objetos y uno de ellos es el de la herencia.

El plan trabajo se dividió básicamente en seis partes que nos ayudarán en la conceptualización del mismo:

I ANTECEDENTES

Aquí se habla en términos generales del lenguaje OBOCCAM y la teoría que lo envuelve, por ejemplo, la programación concurrente orientada a objeto, las arquitecturas paralelas utilizando el procesador TRANSPUTER y OCCAM, las aportaciones del trabajo y la comparación con otros trabajos.

II ARQUITECTURAS DE COMPUTADORAS Y COMUNICACION ENTRE OBJETOS

Se explica brevemente sobre los distintos tipos de computadoras, así como las características técnicas del procesador TRASPUTER y los distintos mecanismos de comunicación para la distribución de procesos.

III EL LENGUAJE CONCURRENTE OCCAM

Se presenta el lenguaje de programación concurrente OCCAM, su filosofía de programación, sus principales proposiciones de programación que son utilizadas en la construcción de procesos concurrentes.

IV PROGRAMACION ORIENTADA A OBJETOS EN OCCAM

En el capitulo IV se explica la implementación de los objetos utilizando OCCAM, el esquema utilizado para representarlos, así como la solución seleccionada para su desarrollo.

V OCCAM BASADO EN OBJETOS

Aquí se describe el modelo de programación del lenguaje OBOCCAM, presentando la extensión de sus nuevas palabras reservadas, la implementación de los conceptos empleados en el desarrollo del lenguaje como son la definición del objeto, su declaración, sus principales componentes, entre otras cosas.

VI APLICACIÓN Y DESARROLLO

Finalmente validaremos el uso del lenguaje OBOCCAM, con unos ejemplos en los se muestran sus atributos.

CAPITULO I

ANTECEDENTES

En este capítulo definiremos los fundamentos que involucran a OBOCCAM, para que al término del mismo sea evidente el objetivo y la justificación del trabajo.

Primeramente veremos los principales conceptos de la programación concurrente y la orientada a objetos que son involucrados en el desarrollo de la extensión del lenguaje OCCAM, al igual que se analizan algunas características importantes de las arquitecturas paralelas y en esencia, aquellas que se ven involucradas con el TRANSPUTER, al final del capítulo se muestra la justificación del trabajo y los objetivos logrados, además de una breve comparación con otros trabajos que hasta el momento se tiene conocimiento de ellos.

I.1. Introducción

En los distintos caminos que ha tomado la computación, se han presentado nuevas arquitecturas de cómputo, metodologías y herramientas, mediante las cuales podemos combinar y construir alternativas para obtener nuevas soluciones a los problemas que se han presentado, las cuales son cada vez más optimas y cercanas a las exigencias del mundo de hoy.

Conforme profundicemos en el trabajo, entenderemos tanto el enfoque, como los objetivos que perseguimos, el resultado final es el lenguaje de programación concurrente basado en objetos llamado OBOCCAM el cual en términos generales lo definiremos con las siguientes palabras:

OBOCCAM : ***OCCAM BASADO EN OBJETOS*** es un lenguaje de programación, el cual es una extensión del lenguaje de programación concurrente OCCAM.

El lenguaje OCCAM es ejecutado en una computadora conformada por procesadores TRANSPUTER[1][5] de 32 bits con capacidad de interconectarse entre ellos para formar redes de procesadores y explotar el cómputo paralelo.

Las principales áreas con las que se vincula el trabajo son :

- La programación concurrente.
- La programación orientada a objetos.
- Las arquitecturas de cómputo paralelo.
- El procesamiento paralelo utilizando el procesador TRANSPUTER y OCCAM [12].

[1] En la bibliografía, aparecen referencias que orientan al lector al respecto.

Estos son relacionados con la programación concurrente orientada en objetos, la cual día a día da nuevas alternativas como :

- Combinar las técnicas de programación concurrente con la orientación a objetos[7].
- Brindar alternativas que aumentan la capacidad de cómputo
- Reducir tiempo de desarrollo
- Reducir los costos de sistemas de cómputo con alto procesamiento[2]
- Dar nuevos enfoques de análisis, diseño y programación de sistemas.
- Brindar herramientas para el desarrollo de sistemas de tiempo real.

I.2. Entorno de programación

Dentro del entorno de programación del lenguaje OBOCCAM se encuentran técnicas de programación concurrente, ya que el lenguaje OCCAM del cual se deriva, por naturaleza tiene primitivas de programación que permiten la concurrencia de procesos, además, estas primitivas, el procesador TRANSPUTER las implementa a nivel de hardware.

Pero el enfoque de la tesis no es la utilización del lenguaje OCCAM, ni la de sus primitivas de programación concurrente, sino extender su conjunto de proposiciones, para que nos permita la utilización de objetos, esto es, el lenguaje OCCAM utilice las técnicas, que hoy se conocen como la programación concurrente basada en objetos, la cual se define como:

La metodología de diseño y programación, mediante la cual, el sistema construido es modelado mediante una colección de programas (encapsulación[3] de datos e instrucciones) que son ejecutados concurrentemente, denominados objetos que interactúan entre si por medio de paso mensajes.

La motivación de esta investigación, proviene de la necesidad de construir poderosos y flexibles entornos de desarrollo de software paralelo, que reúnan las crecientes demandas para resolver problemas complejos, además de proveer un conjunto de proposiciones para el desarrollo de servicios cada vez más perfeccionados.

Por otra parte cada día aparecen sistemas de cómputo con características excepcionales como el procesador TRANSPUTER, que permiten crear grandes sistemas de procesamiento paralelo a bajo costo y desarrollar sistemas cuyo procesamiento sea distribuido y paralelo.

---

[2] Entiéndase a este tipo de sistemas como aquellos que nos permiten procesamiento vectorial como la computadora CRAY ó Sistemas paralelos como la N-CUBE de Caltech.

[3] Es la propiedad en la cual se oculta el comportamiento interno de las entidades externas.

Si aunamos a las características de la computación paralela, las bondades que nos brinda la programación orientada a objetos(POO)[19] consolidamos la metodología y se incrementan todos los beneficios como:

- Abstracción del mundo real
- Modularidad
- Reutilización de código
- Encapsulamiento
- Reducción de tiempo de programación
- Incremento en la productividad del programador
- Fácil mantenimiento y adaptabilidad a los cambios del negocio
- Incremento en la calidad de las aplicaciones

Lo anterior nos lleva a tener un entorno de programación con objetos, que se encuentren ejecutándose concurrentemente en un solo procesador o que a la vez se ejecuten paralelamente en diferentes procesadores TRANSPUTER conformados en una red.

También se tendrá que disponer de características en las cuales nuestro entorno de programación (OBOCCAM) manipule la comunicación entre objetos concurrentes, que se encuentran dentro de un mismo procesador ó distribuidos aleatoriamente en una red de ellos.

El mismo entorno debe de permitir la abstracción de objetos ó la declaración de sus principales mecanismos como la definición de tipos abstractos de datos, sobrecarga de operadores, polimorfismo, declaración de métodos del objeto, entre otros.

Similarmente deben resolverse mecanismo que nos permitan explotar los servicios otorgados por los objetos esto es, que se utilice el llamado a objetos remotos[4].

I.3. Programación concurrente basada en objetos y arquitecturas paralelas

Conceptualmente la explotación del paralelismo es atractivo, ya que nos brinda un conjunto de proposiciones de programación más amplio para la solución de problemas, aunque no es fácil aplicarlo ya que se requiere de más experiencia y de amplios conocimientos en la construcción de algoritmos computacionales.

Actualmente se han instrumentado muchas técnicas de programación concurrente[5] en disciplinas tradicionales como son los sistemas operativos y las bases de datos distribuidas[7], pero estas técnicas son insuficientes, ya que en el

[4] Se relaciona con la propiedad de poder interactuar con otros objetos y requerir los servicios que estos ofrecen.

[5] En casi todo el trabajo se utilizan términos concurrencia y paralelismo, aunque el paralelismo es un caso especial de la concurrencia, aquí los diferenciamos desde el punto de vista de ejecución, ya que únicamente existirán objetos paralelos cuando existan múltiples procesadores.

análisis, diseño e implementación de programas paralelos se requiere amplias variantes de sus componentes así como un alto grado de concurrencia.

En el diseño de programas que utilizan técnicas de paralelismo, existe el problema de la partición de los distintos componentes, así como, la definición del paralelismo y su función.

Una alternativa que se da para el análisis, diseño e implementación , es la utilización de metodologías orientadas a objetos, con ellas podemos disponer de nuevas formas de pensar para el desarrollo de procesos paralelos.

La transición de un proceso a un objeto es natural, ya que al analizar detenidamente los procesos como módulos de programas y datos, podemos abstraerlos fácilmente a objetos, mediante el encapsulado de datos y programas como un ente concurrente, el cual mediante los servicios necesarios nos daría su comportamiento.

Idealmente OBOCCAM nos daría todas las bondades de orientación a objetos, aunque realmente se ha limitado su funcionalidad, como la de no tener mecanismos de herencia.

En los lenguajes concurrentes orientados a objetos se han desarrollado mecanismos de herencia, los cuales, al implementarlos como parte del lenguaje han resultado complicados, además, otro factor por lo que no la hemos considerado, es que difícilmente se puede mantener un control, cuando se utiliza en declaraciones múltiples y cuando existen declaraciones en las que se tienen varios niveles jerárquicos de herencia.

En un lenguaje orientado a objetos puro, uno de sus principales atributos es la herencia, es por esto que a falta de esta característica se le denomina basado en objetos y no orientado.

La programación orientada a objetos incluye conceptos tales como : clases, objetos, tipos de datos abstractos, mensajes, herencia y polimorfismo, donde uno de los principales conceptos es las Clase.

Una clase es un conjunto de objetos con las mismas características, dentro del lenguaje de programación, esta clase es la declaración del tipo de dato definido por el usuario: un tipo abstracto de datos.

La clase puede adoptar características definidas en otra clase, esto es la herencia.

La herencia siempre se encuentra ligada a la clase o al tipo de dato abstracto, en cambio un objeto simplemente aparecerá como una entidad perteneciente a la clase.

El Objeto al igual que la clase es definido por el usuario como un tipo de dato que no soporta la herencia, como en el caso de OBOCCAM.

Para OBOCCAM el concepto herencia no existe[6] y en su lugar para declarar un objeto es por medio de la definición de un tipo de dato definido por el usuario, al cual se le llama simplemente OBJECT y a partir de este se declaran todas las instancias que haya diseñado el programador.

Un objeto concurrente básicamente es la Encapsulación de datos y procedimiento, los cuales pueden ser privados o públicos.

Los únicos autorizados para accesar la información privada de los objetos son sus propios procedimiento o alguna función amiga como en el caso de C++

El polimorfismo es la habilidad de un método para poder reaccionar a diferentes llamados, en otras palabras es el mecanismo para que en tiempo de ejecución existan métodos nombrados de la misma forma pero que reaccionan de diferente forma dependiendo del objeto que los invoca.

Otro caso de polimorfismo es la sobrecarga de operadores, donde un operador(+, -, *, /, etc.) reacciona de forma diferente dependiendo de los operandos que se encuentran involucrados en la operación.

Los mensajes son solicitudes enviadas por un objeto para el servicio/asistencia de un segundo objeto, además es uno de los principales mecanismo que ofrece OCCAM para la comunicación entre procesos.

OBOCCAM explota los conceptos antes mencionados, brindando la abstracción del mundo real como objetos interactuantes entre sí, por medio de paso mensajes y ejecutándose concurrentemente.

Los lenguaje para la programación concurrente basada en objetos tienen que proveer construcciones de lenguaje que le permitan al diseñador/programador expresar los componentes del sistema tan directamente como le sea posible y además tener la habilidad para poder expresar detalladamente el más perfecto algoritmo.

Además facilitan la construcción de robustos sistemas de software bien estructurados, sin embargo no debemos olvidar que se necesitan esquemas de depuración, potentes para modelos individuales de programación paralela los cuales pueden reforzarse por medio de estas técnicas.

---

[6] Aunque una extensión del trabajo es poderle dar mecanismos a OBOCCAM para que soporte esta característica y logre ser un lenguaje orientado a objetos.

I.4. OCCAM y el TRANSPUTER

El lenguaje de programación OCCAM nos brinda un conjuntos de facilidades para explotar la concurrencia en los procesadores TRANSPUTER algunas son.

- Se deriva del modelo matemático conocido como CSP implementado por Hoare[10].

  El desarrollo del TRANSPUTER y su lenguaje nativo OCCAM respectivamente, fue fundamentado con los mecanismos descritos por C.A.R Hoare en su articulo llamado Communicating Sequential Processes, en el cual se plantea, un conjunto de primitivas de programación paralela.

  Básicamente OCCAM se basa en estas primitivas y el TRANSPUTER proporciona los mecanismos a nivel de hardware para optimizarlas.

- Comunicación entre procesos por medio de paso de mensajes[9].

  El mecanismo básico para la comunicación entre procesos es el envío y recibo de mensajes, los cuales son paquetes definidos que ayudan a sincronizar los procesos para la realización de una tarea.

- Construcciones para la manipulación de entradas y salidas.

  Los mensaje son recibidos y enviados a través de variables que se declaran de un tipo especial llamado canal, el cual nos auxilia en la definición de un medio de comunicación entre procesos.

- Construcciones para el manejo de procesos paralelos.

  OCCAM cuenta con instrucciones que nos especifican la concurrencia, lo que significa que todos aquellos procesos que se encuentran definidos dentro de estas instrucciones serán ejecutados en paralelo.

- Replicación de procesos.

  La replicación de procesos es un mecanismo que nos ayuda a tener un conjunto de procesos similares a partir de uno definido con anterioridad.

- Manipulación de comunicación selectiva.

  La utilización de canales nos permite la comunicación síncrona entre dos procesos, aunque por las características de las primitivas de comunicación, un proceso se bloquea cada vez que espera el mensaje liberado por otro, lo que implica, que si existen varios procesos esperando alguna respuesta de un proceso bloqueado, no podrán continuar su ejecución hasta que este reciba el mensaje esperado y continué atendiendo las peticiones de los otros procesos.

Una forma de evitar lo anterior, es utilizando construcciones de comunicación selectiva, la cual nos ayuda a seleccionar un canal de varios, con la condición de que en este canal se presento el arribo de un nuevo mensaje, de lo contrario si en ninguno de los canales se presento algún arribo, la secuencia del programa continua.

- Utiliza el concepto de guardias implementados por Dijkstra

  Este mecanismo se combina con la comunicación selectiva y significa la utilización de variables booleanas llamadas guardias que son asociadas por cada canal de arribo de mensajes, esto es, cuando en algún canal arriba un mensaje y su variables asociada es verdadera, entonces se procede a atender el arribo del mensaje de lo contrario, si el guardia asociado es negativo, entonces no se procede a la lectura del canal.

- Construcción para la declaración de protocolos de mensajes entre procesos.

  OCCAM nos permite la declaración de protocolos que son mecánismos que nos ayudan a definir formatos predefinidos, los cuales son utilizados para definir canales que soportaran mensajes construidos en base al formato.

  En el capítulo III extenderemos estas construcciones, en el caso de que sea necesario conocer más de las distintas construcciones propuestas por Hoare, entonces recomendamos la lectura del articulo llamado Communicating Sequential Proccess[10]

Desde el punto de vista de la arquitectura de la computadora utilizada para el desarrollo del lenguaje, nos encontramos al procesador TRANSPUTER, el cual a continuación se resumen algunas de sus características[3]:

- Están construidos bajo una filosofía RISC.
  La arquitectura básica del procesador utiliza un conjunto de instrucciones reducido y toma algunas características de este tipo de procesadores[3].

- Son procesadores de 32 bits.
  El tamaño de la palabra del procesador, el bus de datos y el de instrucciones es de 32 bits

- Tiene cuatro enlaces de comunicación para interconectarse con otros cuatro procesadores.
  Cada procesador TRANSPUTER cuenta con cuatro canales bidireccionales que son utilizados para interconectarse con otros cuatro procesadores y formar redes con diferentes topologías que nos ayuden a explotar el paralelismo.

- Su velocidad de comunicación es de 10Mbits por segundo.
  La comunicación entre los procesadores es de 100Mbits/s, el cual utiliza para sus operaciones de entrada y salida ciclos de DMA[7].

- Tiene implementado a bajo nivel, la mayoría de las primitivas para la manipulación de la concurrencia.
  Muchas de las instrucciones que utilizamos en la manipulación de la concurrencia, son implementadas a nivel de hardware, con lo que se pretende maximizar la velocidad de los procesos que requieran de estas construcciones.

- Su lenguaje de programación a bajo nivel es OCCAM.
  Uno de las mejores contribuciones que se hicieron fue la de mantener como lenguaje nativo para la programación del TRANSPUTER a OCCAM.

- Cada procesador tiene una memoria.
  Como se explica en el capítulo II, cada procesador cuenta con su propia memoria y la única forma de compartir información entre procesos es por paso de mensajes.

La evaluación general de las tendencias de estas arquitecturas se ven reflejadas en el desarrollo de los nuevos procesadores TRANSPUTER los cuales nos brindan nuevas características y evolucionan tanto en arquitectura como en el lenguaje OCCAM ofreciéndonos nuevos métodos para la explotación del cómputo paralelo y distribuidos.

Con OBOCCAM damos una interface diferente para el desarrollo de sistemas que se ejecutan sobre una red de procesadores TRANSPUTERS, extiende homogéneamente el lenguaje OCCAM y reúne la programación concurrente y la programación orientada a objetos para que así se tenga una nueva herramienta de desarrollo y puedan realizarse incursiones en una área de la computación conocida como programación concurrente orientada a objetos.

I.5. Objetivos y justificación del trabajo

El objetivo es la extensión del lenguaje de programación concurrente llamado OCCAM, con un conjunto de nuevas construcciones que nos van a permitir la programación basada en objetos, a lo que le hemos llamado OBOCCAM.

Para generar un lenguaje concurrente orientado a objetos tenemos cuatro alternativas:

- Diseñar un nuevo lenguaje concurrente orientado a objetos.
- Extender un lenguaje existente orientado a objetos y darles algunas características para la gestión de la concurrencia.
- Crear un conjunto de librerías a un lenguaje orientado a objetos y por medio de estas, la forma de implementar la concurrencia.

---

[7] Acceso directo a memoria.

- Extender un lenguaje concurrente y darle características de orientación a objetos.

En nuestro caso seleccionamos la ultima alternativa, El lenguaje OCCAM concurrente, lo extendimos con nuevas sentencias con la finalidad de manipular objetos.

Las ventajas que nos ofrece esta selección son:

- Aprovechar los mecanismos para la gestión de la concurrencia proporcionados por el lenguaje OCCAM.

- Explotar óptimamente la arquitectura del procesador paralelo ya que OCCAM y OBOCCAM se desarrollaron dentro del mismo paradigma.

- El trabajo se orienta hacia la utilización de las arquitecturas paralelas basadas en TRANSPUTER y los lenguajes de programación concurrente orientados a objetos.

Además, tomando en consideración otras alternativas en el diseño del lenguaje, creamos un conjunto de librerías para la comunicación entre objetos y facilitamos la interacción entre ellos y sobre todo la factibilidad del llamado a funciones que ofrecen objetos que se encuentran potencialmente en ejecución remota.

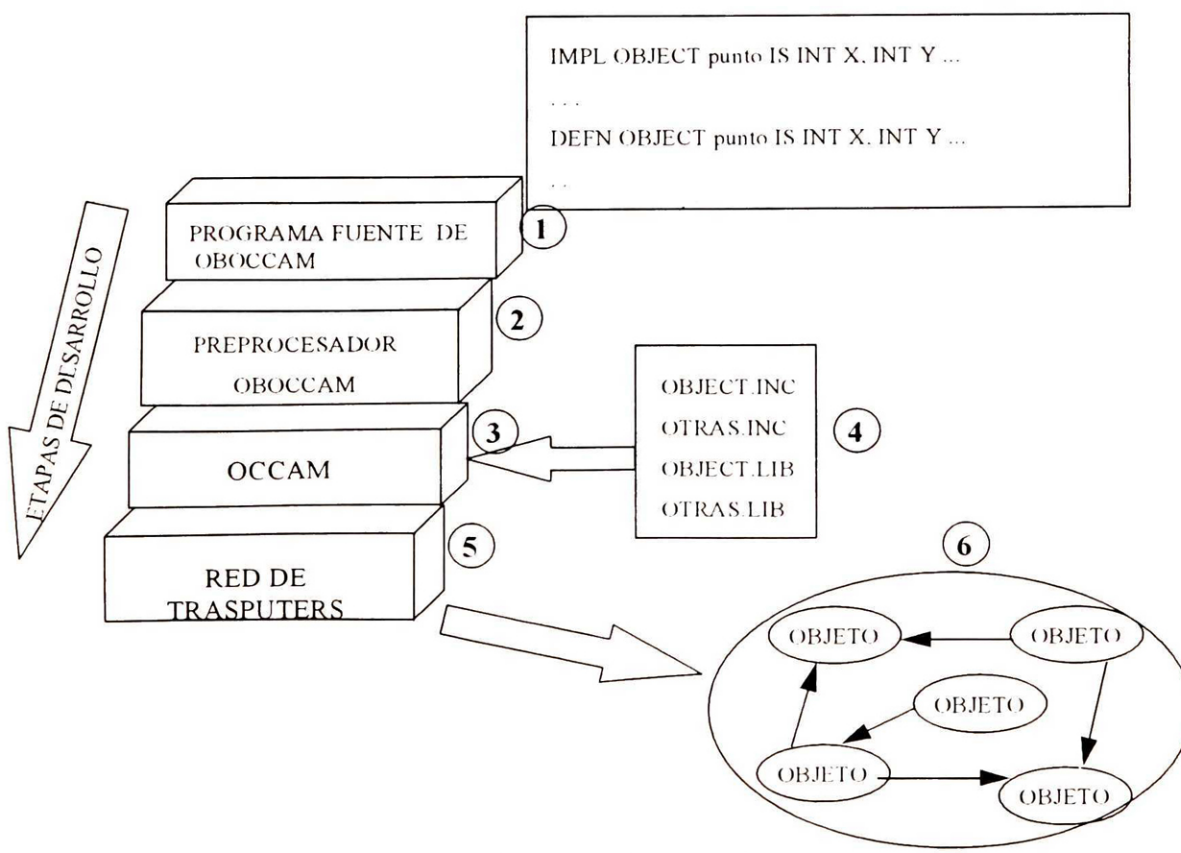


Figura 1. Sobrevista de OBOCCAM

La figura 1 nos detalla los pasos que se siguen en el desarrollo de un programa de OBOCCAM los cuales se resumen en los siguientes pasos:

1.- Programa fuente del lenguaje OBOCCAM.
Este es un archivo de formato texto generado por el programador y contiene el código fuente del programa.

2.- Preprocesador de OBOCCAM.
El preprocesador se encarga de transcribir el código fuente a sentencias de OCCAM.

3.- Compilador de OCCAM.
Una vez transcrito el programa fuente este se compila y se obtiene el programa objeto.

4.- Librerías necesarias para la construcción del programa.
Antes de ejecutarse en la TRANSPUTER, el código objeto se enlaza con las librerías necesarias que contengan el código de las sentencias que se utilizan dentro del programa.

5.- Ejecución en un solo procesador o en una red.
Al final el programa se ejecuta en un procesador TRANSPUTER o en una red de ellos.

6.- Vista de la interacción entre procesos.
Todos los objetos interactúan entre ellos por medio de paso de mensajes hacia un objetivo común.

Al final del proceso descrito anteriormente, la ejecución del programa puede llevarse a cabo sobre un solo TRANSPUTER o un conjunto de ellos, para este proyecto en especifico, el desarrollo del preprocesador[8] se empleo el lenguaje C++ de Borland, las pruebas con el TRANSPUTER se utilizo la versión de OCCAM 2 y un Monoputer (Un solo procesador TRANSPUTER) del tipo T805 el cual se encuentra interconectado con una PC, esta se utiliza como el medio para la carga del programa en el procesador y el manejo de las entradas y salidas.

A continuación enumeramos los principales módulos de la extensión del entorno de e OCCAM para obtener así el entorno OBOCCAM.

- Arquitectura abierta cliente-servidor con aplicaciones independiente del medio comunicación.
Significa que se podrán generar aplicaciones cliente, las cuales solicitaran servicios a otros objetos llamados servidores, el protocolo de comunicación entre

[8] Este programa se encarga de realizar toda la tarea de análisis y detectar las construcciones validas de OBOCCAM y transformarlas a su equivalente en OCCAM.

objetos es transparente para el desarrollador, el cual simplemente empleará una sentencia de llamado a un método, por ej. Objeto.metodox(par1,par2).

- Una extensión consistente con OCCAM
  Esto nos indica que la extensión realizada a OCCAM lleva su misma filosofía, además, sería muy natural para el programador utilizar las nuevas construcciones para la manipulación de objetos y su interacción por medio de mensajes.

- Características consideradas de C++: estructuras de datos, encapsulamiento (TAD), polimorfismo, constructores, destructores.

OBOCCAM tiene básicamente las siguientes extensiones:

a) Estructura de datos.- El lenguaje OCCAM como tipo de datos más complejo que brinda es el arreglo, así que, para ofrecer la declaración de objetos tuvimos primero que definir el tipo conocido como estructura de datos.
b) El objeto.- Una vez definida la declaración de estructuras de datos, se hicieron los cambios para la manipulación de tipos abstractos de datos(TDA), el cual es el mecanismo para la declaración de objetos.
c) Se consideraron los casos para la inicialización y terminación de objetos esto es la forma de definir constructores y destructores dentro de las declaraciones de objetos.

El manejo de polimorfimo se incluye al permitir la declaración de los métodos de los objeto y la sobrecarga de operadores.

- Administradores de objetos pasivos (servidores) los cuales reciben los mensajes (operación).
  La declaración de objetos nos permite manipular objetos que se encuentran ejecutándose y que al arribo de un mensaje, los objetos respondan a él, ofreciendo algún servicio determinado.

- Objetos remotos (llamadas a métodos que ofrecen objetos remotos).
  La abstracción por medio de un administrador de mensajes el cual nos permite utilizar servicios de objetos que se encuentran ejecutándose en algún otro procesador o tal vez en el mismo.

- Desarrollo de aplicaciones por capas jerárquicas.
  La posibilidad de mantener técnicas de programación en las que nosotros creemos servicios, que sean utilizados por otras funciones más complejas y estas a su vez, ofrezcan otros servicios a capas superiores y así tener librerías de código reutilizable(componentes).

También se mencionan otros puntos que se incluyen en este trabajo y que se consideran importantes

- Un conjunto de teoría relacionada con los TRANSPUTER y las arquitecturas del tipo MIMD[1].

- Una exposición de los conceptos relacionados con el lenguaje de programación OCCAM.

- El conjunto de conceptos utilizados para la realización del trabajo.

- Un ejemplo que muestra los conceptos fundamentales de OBOCCAM.

- Un estatus general de los lenguajes concurrentes orientados a objetos.

- La implementación del preprocesador con las siguientes características:

a) Puede recibir como entrada un código fuente con construcciones en OCCAM más las nuevas construcciones, que son las que nos permiten utilizar los conceptos de objetos.
Esto es porque, para la realización del proyecto se diseño el preprocesador utilizando todo el conjunto de la gramática de OCCAM, adicionándole las nuevas construcciones.

b) Nos da como salida el código en OCCAM para que sea compilado y cargado en el TRANSPUTER.

c) La posibilidad de extender los algoritmos para que OBOCCAM sea cada vez más completo.

I.6. Lenguajes concurrentes orientados a objetos y algunas diferencias con el presente trabajo.

- ACTOR[7]
Carl Hewitt fue el desarrollador original de este lenguaje, el cual toma como antecesor a los lenguajes Lisp y Simula.

  ACTOR organiza los programas como colecciones de objetos activos que se ejecutan en paralelo y se comunican por medio de paso de mensajes, los objetos son dinámicos y autoconfigurables y presentan herencia. Además permite compartir recursos, OBOCCAM y en esencia el modelo CSP del cual se deriva OCCAM no lo permite.

  Este lenguaje generalmente es utilizado para el manejo de soluciones en inteligencia artificial como la representación por medio de objetos de bases de datos de conocimientos.

- BETA.[7]
Es un lenguaje basado en el lenguaje de programación simula, utiliza el concepto de PATRON, en el cual asume las características de una clase, utiliza procedimientos, funciones y tipos, además, un patrón interactúa con otros Patrones por medio de mensajes.

Ejecuta igualmente corrutinas como procesos, y en su modelo de programación utiliza los conceptos de programación procedural y funcional; no permite múltiple herencia y ofrece algunas características de la programación funcional, este lenguaje fue desarrollado sobre computadoras Macintosh SUN y Sysware Sus.

- VULCAN Logical Concurrent Object [7]
  Este sistema tiene un grado de concurrencia fino, sincronización y encapsulamiento.

  Su desarrollo está basado en los lenguajes lógicos como Prolog concurrente, soporta herencia en base a delegación y utiliza envío y recibo de mensajes a través de colas.

  Utiliza conceptos de comunicación por medio de canales y algunos otros conceptos importantes manejados en la programación de tipo lógica.

- CONCURRENT SMALLTALK [7]
  Incorpora constructores de concurrencia, nuevos mecanismos de sincronización y objetos atómicos, en realidad es el Smalltalk80 incluyendo sólo primitivas para el manejo de concurrencia.

  Crea objetos dinámicamente, la activación de la concurrencia es a través de recibo de mensajes, la comunicación es por medio de mensajes y llamada a procedimientos remotos y su sincronización es por medio de un objeto llamado CBOX el cual se comporta como un MailBox

- ORIENT84/K [7]
  Se utiliza principalmente en inteligencia artificial, propone un modelo para representar el conocimiento como objetos concurrentes, el compilador está codificado en C y básicamente es un lenguaje derivado del Lisp, fue desarrollado para ejecutarse en máquinas con sistema operativo UNIX 4.2BSD sobre plataformas como una VAX y SUN II Workstation, así que, sus primitivas de sincronización y comunicación son similares a las de UNIX.

- POOL-T [13]
  Fue diseñado por Pierre America para describir sistemas que se ejecutan en arquitecturas paralela, utiliza el paso de mensajes síncrono y rutinas, en este lenguaje también se excluye la herencia, al igual que en OBOCCAM emplea el concepto de envió y recibo de mensajes a través de canales.

- ABCL/1 [13]
  En este lenguaje, la creación de objetos es dinámica al igual que la relación entre objetos es modificable en tiempo de ejecución, la comunicación es asíncrona por paso de mensajes y utiliza buffers en el recibo, emplea herencia por medio del mecanismo de delegación, este lenguaje es funcional como Lisp y no posee mecanismo de tiempo real.

- OBOCCAM
  Se comunica entre procesos a través de paso de mensajes sincronos, su esquema de sincronización es a través de bloqueo, no permite la construcción dinámica de objetos, ni comparte variables o recursos con otros objetos, se deriva básicamente del modelo matemático CSP(Communicating Sequential Process de Hoare) se ejecuta en redes de procesadores TRANSPUTER, y utiliza el "send" y "receive" para la manipulación de mensajes y para especificar el paralelismo utiliza el "Cobegin"/"Coend" los cuales indican el inicio y el final de código que será ejecutado en paralelo.

Existen otros lenguajes concurrentes orientados a objetos, con distintas filosofías y modelos computacionales, aquí solo se muestran algunos de ellos para que se tenga una comparación general mediante sus paradigmas[9]

En resumen :

Lenguaje	Diseñador	Paradigma	Nombre del mecanismo de abstracción	Sincronización y Comunicación	Permite herencia	Comparte Recursos	Permite creación dinámica
Actor	Carl Hewitt	Funcional	Objeto	Paso de mensajes asíncrono	Si	Si	Si
Beta	Bent Broun Et al	Funcional y procedural	Patrón	Paso de mensajes	Si	Si	Si
Vulcan	Kernet Kant Et al	Lógico	Objeto	Paso de mensajes a través de colas	Si	Si	- - -
Concurrent SmallTalk	Yakote and Tokoro	Funcional	Clase	Paso de mensajes	Si	Si	Si
Orient/84	Yutaka Ishikawa	Funcional	Objeto	Variables compartidas	No	No	- - -
Pool-T	Pierre América	Procedural	Objeto	Paso de mensajes	No	- - -	No
ABCL	Akinore yonezawa Et al	Funcional	Objeto	Paso de mensajes síncrono	Si	- - --	Si
OBOCCAM	R.Hdez O.Olmedo	Procedural	Objeto	Paso de mensajes síncrono	no	No	No

I.8 Conclusiones

Hemos explicado los mecanismos que se han seleccionado en la extensión, los cuales en parte se han tomado del lenguaje de programación de C++, estos mecanismo los hemos considerado de gran importancia, ya que nos coloca a un paso de lo que es la programación concurrente orientada a objetos, sin olvidar que debemos de enriquecerlo para tener un producto entorno más evolucionado, para ello, se deben considerar otros mecanismos como son la herencia, la persistencia, la creación dinámica de objetos, así como la definición de modelos de comunicación de objetos distribuidos.

---

[9] Básicamente se distinguen en la actualidad cuatro paradigmas en los lenguajes de programación:
a) Programación lógica.
b) Programación funcional.
c)Programación procedural
d)Programación orientada a objetos

El objetivo primordial de este primer capítulo es el de introducir al lector a la teoría que envuelve al TRANSPUTER y su lenguaje de programación OCCAM, además de la explicación de los mecanismos seleccionados para la extensión del nuevo lenguaje OBOCCAM, no perdamos de vista que en los siguientes capítulos conoceremos la arquitectura del TRANSPUTER, la programación de OCCAM y los algoritmos que se implementaron para la construcción del preprocesador OBOCCAM.

CAPITULO II

ARQUITECTURAS PARALELAS Y COMUNICACION ENTRE PROCESOS

II.1. Introducción

En el capítulo anterior se tocaron puntos importantes del trabajo para la implementación del preprocesador que nos ayuda a la transcripción del código de OBOCCAM a OCCAM.

Aquí se presentan los conceptos relacionados con arquitecturas paralelas, su clasificación, el procesador TRANSPUTER y los factores más importantes en la comunicación entre objetos.

Este último tema es de gran importancia, principalmente porque una de las principales aportaciones se centra en la comunicación entre objetos remotos que solicitan servicios otorgados por otros objetos que se ejecutan en el mismo o en otros procesadores.

II.2. Arquitecturas paralelas [1][3]

Normalmente en los sistemas secuenciales se tienen limitaciones inherentes, principalmente no se puede incrementar indefinidamente la velocidad de los circuitos, ya que la mayoría de los transistores están ociosos la mayor parte del tiempo.

Todos los intentos de resolver el problema se inician con la suposición de que una computadora debe de contar con cierto número de unidades aritméticas y lógicas de control y módulos de memoria que operen en paralelo.

La alternativa anterior nos lleva a tener computadoras con múltiples procesadores que pueden ejecutar un gran número de programas todos ellos independientes entre sí ó programas divididos de tal forma en procesos que cooperativamente se comunican, para la obtención de una meta en común.

Si nosotros tenemos un conjunto de procesos ejecutándose en una misma computadora con varios procesadores podemos denotar dos clases de paralelismo:

- Paralelismo ordinario que ejecuta varios procesos en paralelo, con poca o ninguna comunicación entre sí.

- El paralelismo fino que ejecuta varios procesos en paralelo con una extensa comunicación entre ellos.

Estos sistemas de cómputo paralelo, pueden ser de dos tipos:

- Sistemas de bajo acoplamiento: son aquellos que contienen un pequeño número de CPU interconectados y que su comunicación es de un ancho de banda bajo.

- Sistemas de alto acoplamiento son aquellos en los que sus CPU se encuentran interconectados y existe una alta interacción entre ellos.

En la mayoría de los casos, los problemas con paralelismo ordinario trabajan en sistemas de bajo acoplamiento, mientras que los problemas con paralelismo fino lo hacen mejor en sistemas de alto acoplamiento.

II.3. Clasificación de arquitecturas de computadoras

La clasificación más aproximada de los sistemas en paralelo la creó FLYNN[1] en 1972 y está basada en dos conceptos: flujo de instrucciones y flujo de datos, la primera corresponde a un contador de programa (Program Counter), mientras que la segunda corresponde al número de datos que es capaz de procesar la computadora en un ciclo de reloj.

Un sistema con n cpu tiene n contadores y por consiguiente n flujos de instrucciones, donde n es cualquier número entero positivo.

FLUJO DE INSTRUCCIONES	FLUJO DE DATOS	NOMBRE	EJEMPLOS
1	1	SISD	MÁQUINA DE VON NEUMANN
1	MULTIPLE	SIMD	MÁQUINAS VECTORIALES
MULTIPLE	1	MISD	NINGUNA
MULTIPLE	MULTIPLE	MIMD	MULTIPROCESADORES

Figura 2 Clasificación de FLYNN [1]
S= Simple, I= instrucciones, M= Múltiple, D= Datos

En la figura 2 se aprecia las combinaciones posibles cuando se considera el flujo de instrucciones y de datos las cuales son:

1.- Simple flujo de instrucciones con múltiple flujo de datos (SISD).
La mayoría de computadoras que hasta el momento se han construido pertenecen a este tipo de arquitectura, también se dice que siguen la arquitectura de la máquina de Von Neumann[1] propuesta en 1939.

2,- Simple flujo de instrucciones con múltiple flujo de datos (SIMD).
Este tipo de computadoras se encuentran orientados a procesamiento numérico como lo son las computadoras Cray, IBM-9021, etc. dentro de sus procesadores se encuentran vectores de memoria en donde se almacenan datos que son procesados en un solo ciclo de operación.

3.- Múltiple flujo de instrucciones con simple flujo de datos (MISD).
Prácticamente no existe ninguna computadora de este tipo , es teórica

4.- Múltiple flujo de instrucciones con múltiple flujo de datos (MIMD).
Este tipo de computadoras cuentan con múltiples unidades de procesamiento que son capaces de procesar independientemente los datos almacenados en su memoria, que puede ser privada o compartida por cada unidad de procesamiento.

Los sistemas basados en TRANSPUTER se encuentran clasificados según la clasificación de FLYNN en la clase MIMD (MÚLTIPLE FLUJO DE INSTRUCCIONES MÚLTIPLE FLUJO DE DATOS) como se muestra en la siguiente figura:

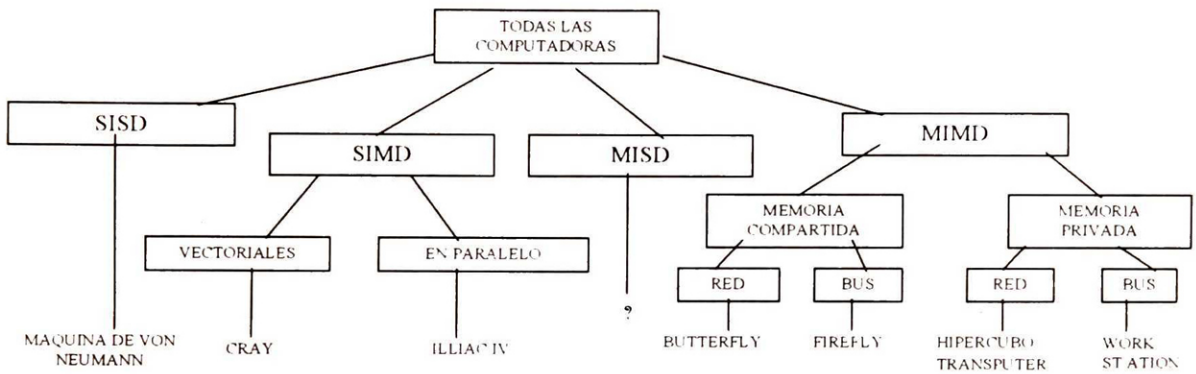


Figura 3 Clasificación de computadoras (según FLYNN [1])

II.4. El procesador TRANSPUTER[3]

Como elementos para la programación de sistemas de cómputo paralelo, utilizaremos en este caso sistemas basados en TRANSPUTER del tipo INMOS; los cuales son sistemas de procesamiento integrados con cuatro canales de comunicación que al irse interconectando con otros componentes de similares características, se forman redes de procesadores que operan en paralelo y permiten la comunicación entre procesos por medio de paso de mensajes.

El TRANSPUTER no es uno de los elementos más representativos de las computadoras RISC, ya que tiene algunas características que no son propias de estas computadoras pero a pesar de esto se les puede clasificar como una computadora del tipo RISC[3].

El primer TRANSPUTER fue desarrollado en 1985 a la que denominaron T414 fue un cpu de 32 bits y 2kbytes de Ram. Este procesador posee un reloj de 20Mhz con el cual alcanza 10 Mips, posee 4 líneas bidireccionales, que sirven de interface de entrada y salida, además de un controlador programable de memoria.

Posteriormente a fines de 1987, empezó la producción del IMST800, que mejoró el diseño anterior, empleando una mejor tecnología para fabricar memoria y así aumentar a 4Kbyte de Ram en el mismo circuito, además de esto, se pudo implantar también en el circuito, un procesador de punto flotante, el cual logra que el T800 pueda alcanzar una velocidad de 1.5 Mflops con sólo 20Mhz.
El diagrama de bloques es:

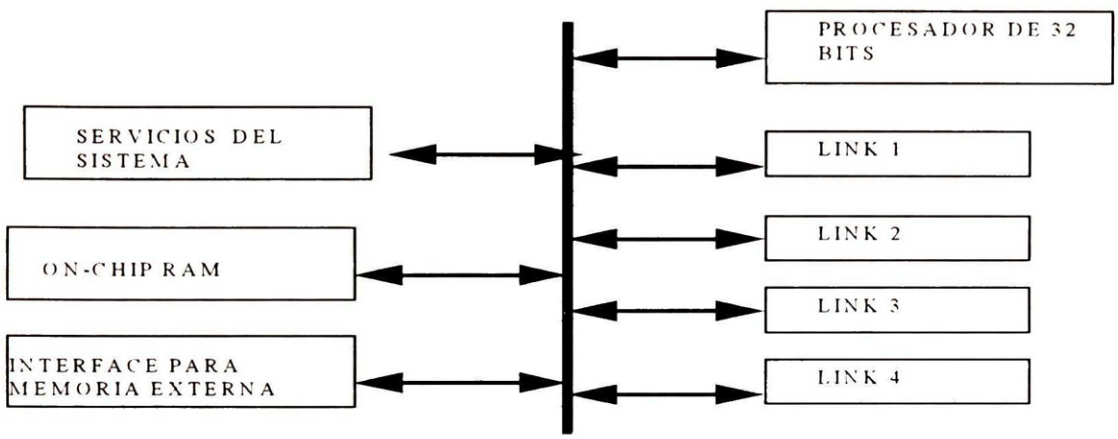


Figura 4. Sistema de interconexión del TRANSPUTER

El TRANSPUTER además del cpu y del bus de datos tiene 4 líneas (LINK) bidireccionales, que sirven para comunicar a este procesador con otros cuatro.

La comunicación y la transmisión de datos entre procesadores, se lleva a cabo usando la técnica de DMA, razón por la cual la comunicación es independiente de la parte de control del TRANSPUTER y de la parte receptora, permitiendo así con esto el acceso a memoria local, sin perturbar la ejecución de los procesos que el TRANSPUTER realiza en ese momento de la comunicación y sin quitarles tiempo de procesamiento. Esta característica del TRANSPUTER origina que la transmisión de datos en un sistema compuesto por varios TRANSPUTER no se vea limitado por el bus de comunicación del sistema.

II.5. Procesamiento paralelo

Definiremos procesamiento en paralelo como la actividad de varias entidades (idénticas o heterogéneas), trabajando conjuntamente hacia una meta común. En computación paralela, las entidades son computadoras o procesadores electrónicos y la meta la ejecución de algunas de las siguientes aplicaciones:

- Simulación de sistemas químicos y biológicos
- Tecnología aeroespacial
- Control de estaciones de energía nuclear
- Reconocimiento de imágenes
- Lenguaje natural, entre otros

El TRANSPUTER es una familia de microprocesadores especialmente diseñados para ser usados como bloques de computación paralela.

La primera importancia es que ahora los sistemas de cómputo de bajo costo del tipo MIMD podría competir con las supercomputadoras vectoriales.

El TRANSPUTER es un procesador cuya principal característica es poseer memoria local o privada. Este procesador, al igual que la computadora HIPERCUBO (N-CUBO) tienen esta característica de memoria.

Sin embargo la HIPERCUBO desarrollada en Caltech fue construida utilizando redes de elementos de procesamiento que podían representar fácilmente las características físicas de varios problemas físicos , en tanto, el TRANSPUTER fue desarrollada en un ambiente local, utilizando el paradigma computacional del paso de mensajes de Hoare (CSP,1978)[10].

Otra característica de el TRANSPUTER, es el diseño de hardware especial para ejecutar eficientemente su lenguaje nativo, el OCCAM y otras primitivas concurrentes incluidas en el lenguaje.

II.6. Comunicación entre objetos

El TRANSPUTER puede comunicarse con cada uno de las otros TRANSPUTER'S en un enlace en serie, bidireccional y de alta velocidad.

La comunicación entre procesos se logra por medio de canales, a su vez los canales se pueden mapear a un determinado enlace físico del TRANSPUTER, esto es parte del mecanismo de configuración.

El mecanismo de configuración se define básicamente como la forma de ubicar un determinado proceso y sus canales a un correspondiente procesador y sus enlaces de comunicación, esto es configurar al procesador para que reconozca los procesos que va a ejecutar y su topología de comunicación

La comunicación es el único medio para que cooperen en la realización de una meta en común, varias entidades que se ejecutan paralelamente en una red de procesadores.

Cada procesador se puede considerar como una computadora, en una red de computadoras podemos encontrar distintos elementos que se comunican entre sí por medio de mensajes, utilizando un conjunto de protocolos ya establecidos.

Lo anterior da como consecuencia, que una red de computadoras se puede definir similarmente por los objetos o procesos que se ejecutan en ellas.

Así, una red de objetos se puede definir, como un conjunto de objetos autónomos interconectados lógicamente que se ejecutan concurrentemente en una red de procesadores.

La forma de comunicación entre ellos es por medio de pasos de mensajes, los cuales, se definen como un conjunto de paquetes de datos que viajan de un objeto hacia otro, estos paquetes siguen ciertas convenciones que se conocen con el nombre de protocolos de comunicación.

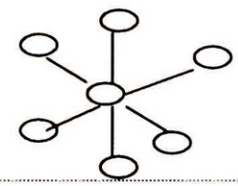
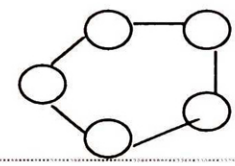
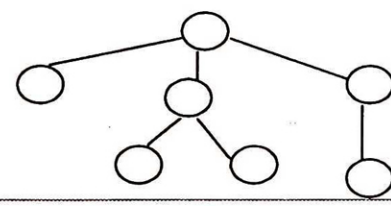
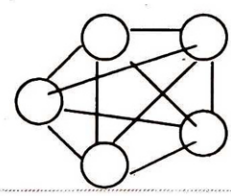
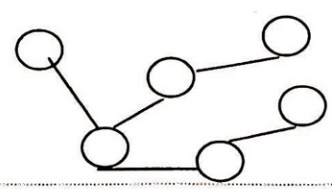
Nombre	Representación
• Estrella	
• Anillo	
• Arbol	
• Completa	
• Irregular	

Figura 5. a) Topologías de interconexión entre objetos

Para OBOCCAM la conexión entre objetos (línea de transmisión) se efectúa por medio de canales de comunicación (CHAN OF) , los cuales son mapeados a enlaces físicos del TRANSPUTER[10]

[10] Considerando que los procesos se están ejecutando en diferentes procesadores.

La figura 5a y la 5b muestra las distintas topologías de objetos que pueden representarse [20]:

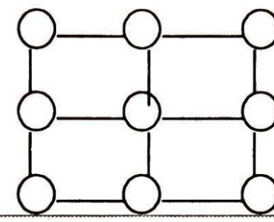
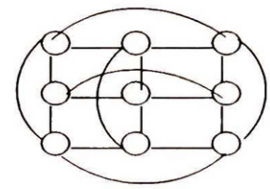
Nombre	Representación
• Matriz	
• Toroidal	

Figura 5. b) Topologías de interconexión entre objetos

En la arquitectura de comunicación entre objetos propuesta para OBOCCAM existe un objeto administrador de mensajes.

La función del administrador de mensajes es similar a la de un redireccionador de mensajes.

El redireccionador de mensajes únicamente envía el mensaje al objeto destino correspondiente, empleando una tabla de direcciones, incluida en su código.

El esquema propuesto en la solución del llamado a objetos remotos, se realizó basándose en el mecanismo que utiliza la capa de transporte conocida como TCP/IP [21].

La forma en que TCP/IP se basa para direccionar una computadora, es identificándola con un número único en la red[11] Por ej. 14.0.0.0 conocido como IP, (internet protocol) cuando una computadora envía un mensaje a otra y no la encuentra en su segmento de red, el mensaje es enviado a un dispositivo (puede también ser alguna computadora) llamado "router"[12] (redireccionador de mensajes Ri donde i=1,2,3...), el cual es el encargado de enviarlo a la dirección requerida, el router tiene una tabla de direcciones para establecer a donde enviará el mensaje cuando el destino no pertenece a la red local.

[11] En nuestro caso será con un número único de proceso.

[12] No es necesario que sea un router también existen dispositivos que tienen el mismo mecanismo pero difieren en el tipo de redes que comunican , a este se le llama gateway.

La computadora para enviar el mensaje utiliza el concepto conocido como subnet mask, el cual es una dirección por defecto y que por medio de la operación X-OR, la computadora logra establecer si se trata de una dirección de la red local o si pertenece a otra. En caso de que se trate de fuera de la red local, directamente la envía hacia el router.

Para ejemplificar el concepto anterior, observe la siguiente gráfica, en especial el dispositivo de direccionamiento R9.

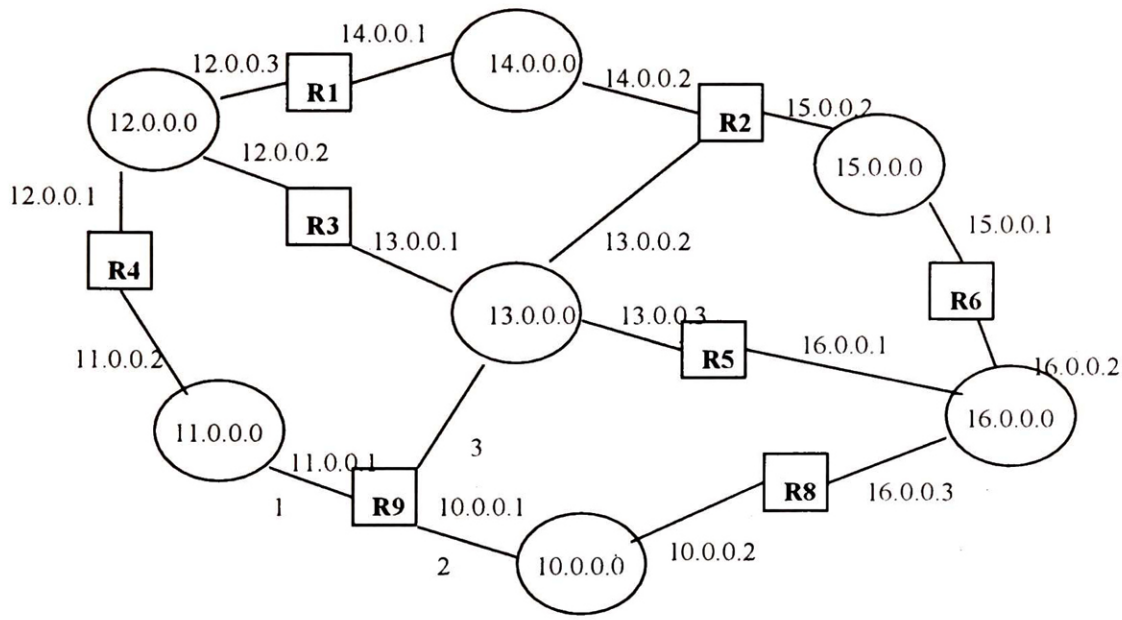


Figura 6. Ruteo de mensajes en red

Cada uno de los círculos nos representan una computadora con una determinada dirección, además tienen ligas que pueden ayudarle a comunicarse con otras computadoras de otras redes, es decir. la liga 1 de la computadora 11.0.0.0.

Dirección de la computadora en la RED	Modo de ruteo	A través de liga
10.0.0.0	Directo	2
11.0.0.0	Directo	1
12.0.0.0	11.0.0.2	1
13.0.0.0	Directo	3
14.0.0.0	13.0.0.2	3
15.0.0.0	10.0.0.2	2
16.0.0.0	10.0.0.2	2

Figura 7. Tabla de ruteo

Por lo tanto la tabla de direcciones del redireccionador R9 que se muestra en la figura 7 puede resolver el arribo de mensajes, por cualquiera de la ligas 1,2,3.

Si el mensaje fuera enviado a otra computadora, al pasar por el redirecionador R9 se revisa la dirección destino y procede de acuerdo a la tabla mediante el algoritmo que se muestra en la figura 8.

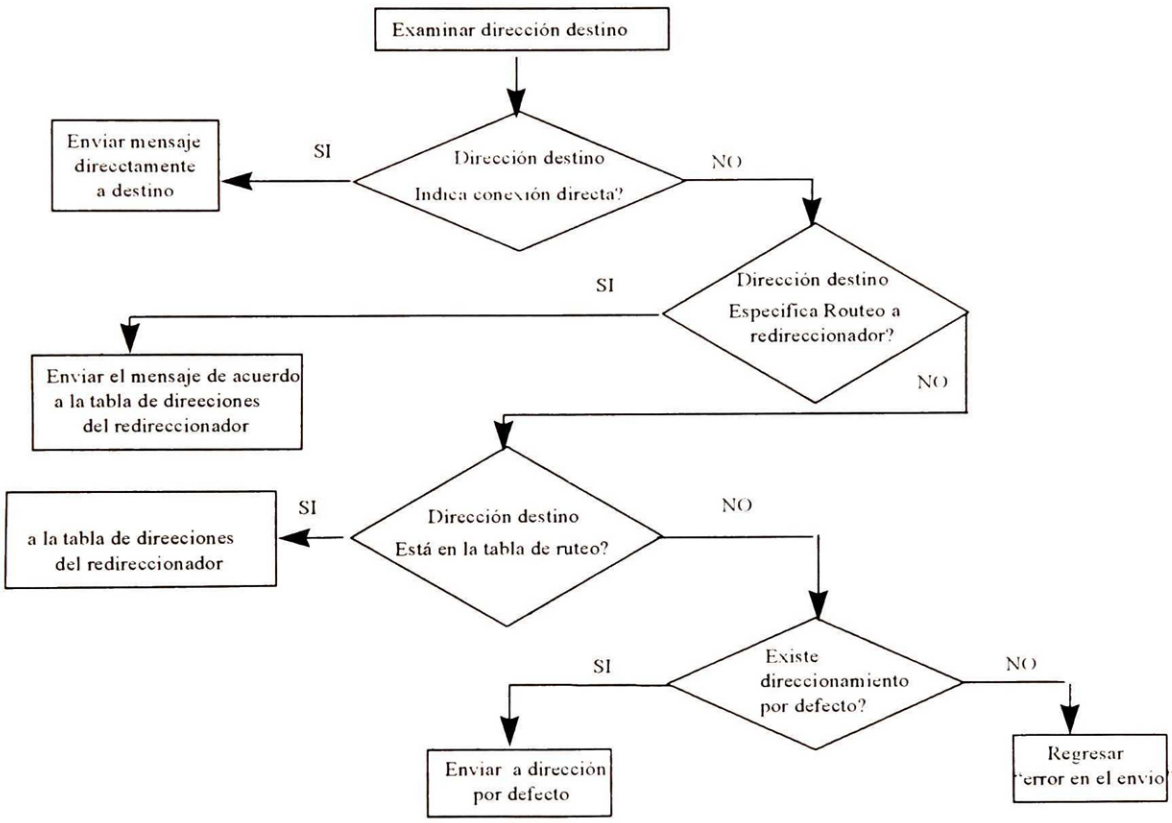


Figura 8. Diagrama del algoritmo de resolución de dirección en una red

II.7. Esquema de comunicación

El mecanismo de comunicación descrito anteriormente, se aplica a la arquitectura propuesta para la comunicación, que se utiliza en el llamado a servicios que ofrecen los objetos remotos, esto es, nos ayuda a resolver el problema de llamar a los servicios que ofrecen algunos de los objetos de la red distribuida independientemente de la topología.

La dirección se resuelve mediante el administrador de mensajes que se muestra en la figura 9.

El administrador de mensajes está conectado a n número de objetos donde n=1,2,3..., los cuales pueden comunicarse entre si a través del administrador.

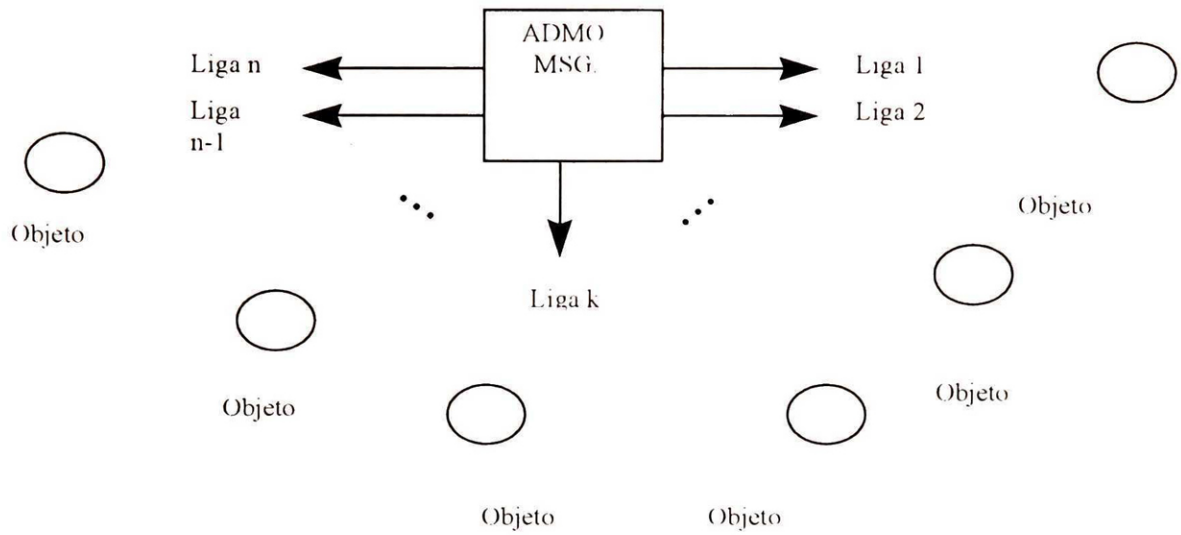


Figura 9. Administrador de mensajes

Para distribuir en la red de procesadores diferentes objetos, un administrador de mensajes puede ejecutarse e interconectarse con otro administrador de mensajes compartiendo su tabla de direcciones para que ambos objetos administradores de mensajes conozcan los objetos que tienen interconectados entre si y así puedan resolver fácilmente el redireccionamiento de mensajes y a su vez la requisición de servicios.

Básicamente la tabla de ruteo se puede construir en base a la topología y al diseño de objetos que realice el programador, para estos existe un conjunto de protocolos que indican el tipo de servicio solicitado.

En la conceptualización de OBOCCAM el administrador de mensajes juega un papel importante, aunque tenemos otros tipos de objetos que interactúan entre sí para proporcionar la funcionalidad del programa en tiempo de ejecución.

En tiempo de ejecución de un programa se tienen dos tipos de procesos :

- Los administradores

  Son los que nos dan la funcionalidad dentro de un programa; en realidad se les ha llamado administradores debido a que ellos llevan el control de los recursos (Memoria, comunicación, llamado a sus procedimientos, etc.) que en tiempo de ejecución son solicitados por las instancias creadas de cada objeto, las cuales son definidas por el usuario. Implícitamente se crea un administrador por cada tipo de objeto que sea declarado.

- Los declarados por el usuario

  En base a las definiciones de objetos, estas son instancias que solicitan servicios a los administradores por ej. las instancia de la definición de un objeto ó una estructura, ó procesos comunes que son programados por el usuario y que no utilizan ninguna de las extensiones que se le hicieron al lenguaje, etc..

  La principal diferencia se observa, en que los objetos administradores no son implementados por el usuario, sino son procesos que son creados en tiempo de preprocesamiento (cuando es analizado por OBOCCAM) y básicamente dan servicios que facilitan las distintas abstracciones escritas en el programa.

  Los tipos de objetos administradores que utilizan nuestra esquema de comunicación son los siguientes:

- De mensajes :
  Se encargan de validar el envío de mensajes hacia los diferentes objetos.

- De objetos definidos por el usuario.
  Gestionan las diferentes instancias de los objetos definidos por el usuario, el código de estos objetos es generado por el preprocesador, además, se crea un administrador por objeto.

- De estructuras de datos definidas por el usuario
  Similar a los administradores de objetos solo que en este caso es para una estructura definida por el usuario, la diferencia recae en el hecho de que la estructura es un objeto que ofrece dos servicios : La consulta y la actualización de algunos de sus datos.

  No olvidemos que el concepto de estructura de dato no la proporciona OCCAM, por lo que es parte de la extensión.

  Los tipos de objetos que utilizan nuestra esquema de comunicación y son definidos por el usuario son los siguientes:

- Procesos del usuario.
  Son procesos que define el usuario, en realidad siguen la sintaxis de OCCAM y no utilizan alguna de las característica de la extensión.

- Administradores de entrada y salida.
  Son procesos que el usuario desarrolla para la manipulación de la entrada y salida de datos

La figura 10 muestra el esquema de comunicación de mensajes entre objetos.

Podemos observar que los objetos, en si mismos, establecen una topología; el problema de direccionamiento es resuelto por los administradores de mensajes,

además vemos que los objetos no necesariamente deben de ejecutarse en un mismo TRANSPUTER.

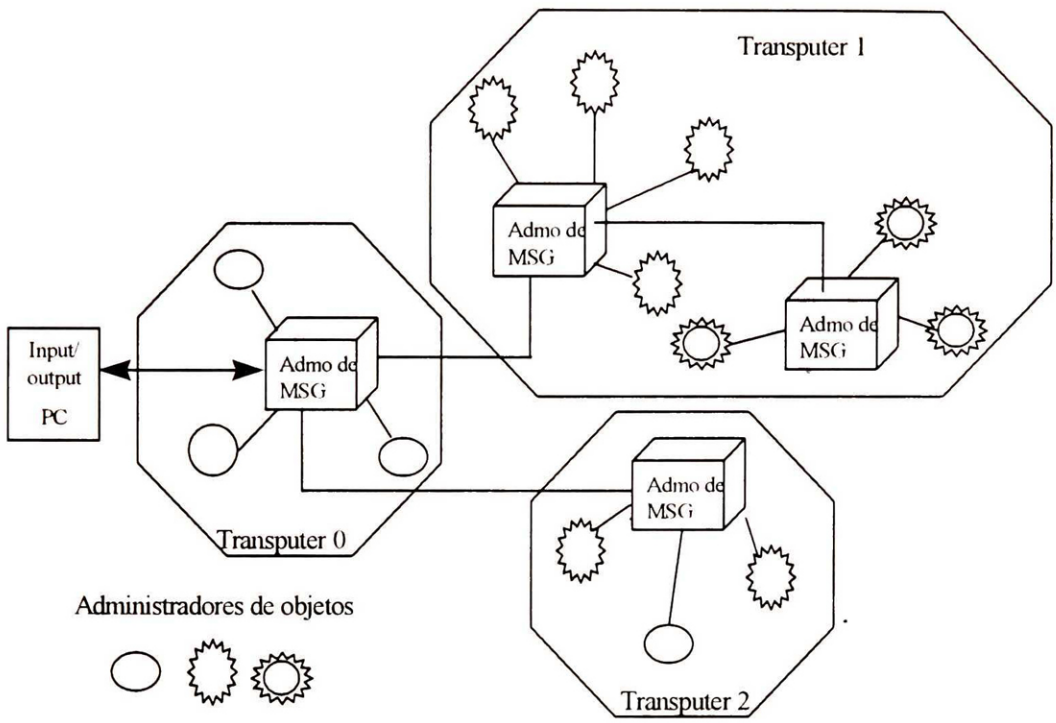


Figura 10. Relación Objetos -TRANSPUTER y administrador de objetos

II.8. Conclusiones

En términos generales se presentó la arquitectura del TRANPUTER, así como, uno de los mecanismos esenciales que se utiliza en la conceptualización e implementación de OBOCCAM: el administrador de mensajes, el cual tiene como objetivo la resolución del ruteo de mensajes entre objetos para la requisición de servicios que estos ofrecen.

No cabe duda que sin el administrador de mensajes no habría la posibilidad del llamado a objetos remotos y con lo cual no se hubiera resuelto este problema.

Al conceptualizar los mecanismos que nos ayudan a resolver los conflictos en tiempo de ejecución y también, ofrecer una capa de abstracción para el manejo de objetos nos da la razón para la implementación de la extensión del lenguaje OCCAM, además de que a nuestros mecanismos se le pueden anexar algunos otros como administradores de memoria, administradores de entradas y salidas etc. que harían mas completo el presente trabajo.

CAPITULO III

EL LENGUAJE CONCURRENTE OCCAM

III.1. Introducción

En el presente capítulo se resumen las características importantes del lenguaje de programación OCCAM[12].

Se observaran básicamente los principales mecanismos en que se basa la filosofía de la programación en OCCAM como son los diferentes tipos de procesos, la implantación de la concurrencia, el manejo de la comunicación, etc.

III.2. Proceso y estructura de un programa.

Una línea de código en OCCAM es un proceso. Así, que es necesario especificar explícitamente la forma en que este se ejecuta. No existen default's dentro de este lenguaje programación.

Un programa, indistintamente del lenguaje, incluye una sintaxis parecida a:

```
DECLARACIONES
INICIALIZACIÓN
PROCESOS
```

Cada una de las líneas anteriores es un bloque que se define de acuerdo a las sintaxis de un lenguaje y que significa :

Declaración : En esta sección se declaran las distintas variables que serán utilizadas en el programa.

Inicialización : Aquí se inicializan las variables a valores determinados para que no existan inconsistencias posteriores.

Procesos: son los algoritmos escritos utilizando las instrucciones que requiramos siguiendo una lógica determinada.

Para indicar una ejecución secuencial de un conjunto de procesos, es necesario la construcción SEQ, por ejemplo :

```
DECLARACIONES
INICIALIZACIÓN
SEQ
   PROCESO.1
   PROCESO.2
   PROCESO.3
```

Esto es, si tenemos 3 procesos que queremos que se ejecuten secuencialmente debemos de anteponer la palabra reservada SEQ lo que le indica a la TRANSPUTER que los siguientes procesos serán ejecutados en forma secuencial.

En este caso PROCESO.1, PROCESO.2, PROCESO.3 nos representan a 3 procesos de OCCAM, el bloque que afecta la palabra reservada SEQ depende de los procesos que estén a continuación, con una indentación de dos espacios.

Los procesos pueden ser anidados, por ejemplo:

```
DECLARACIONES
INICIALIZACIÓN
SEQ
  PROCESO.1
  SEQ
    PROCESO.2
    PROCESO.3
  SEQ
    PROCESO.4
    PROCESO.5
  PROCESO.6
```

Como se observa la anidación de cualquier construcción se representa por la indentación de los procesos, esto es dejando 2 espacios. Las palabras reservadas se escriben siempre con mayúscula y el punto es el separador, aunque en otros lenguajes se utiliza el guión bajo "_" .

El TRANSPUTER no puede desplegar mensajes en pantalla ni comunicarse con el teclado ni al disco. Así, un proceso servidor llamado iserver se ejecuta en la PC en donde reside el TRANSPUTER principal (ó raíz), proviendo estos servicios [5].

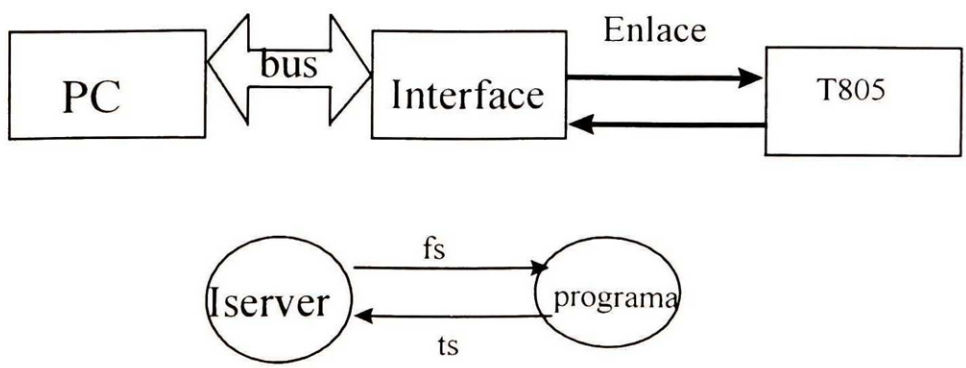


Figura 11 Comunicación entre la PC y el TRANSPUTER

Los canales llamados fs, ts (From.server y To.server), conectan al programa que se ejecuta en el TRANSPUTER al iserver, emplean el protocolo SP (Server.protocolo) que viene definido en las librerías de OCCAM y tienen que definirse antes de referenciarse y son obligatorios.

III.3. Variables, tipos de datos, asignación y constantes[12].

Los nombres de las variables se escriben indistintamente en mayúsculas o minúsculas siempre y cuando no sean palabras reservadas, los nombres de las variables siempre inician con un caracter alfabético, y consisten de una cadena de caracteres alfanuméricos incluyendo el punto, OCCAM es sensible a la forma en que se escribieron, es decir, siempre de que se trate de la misma variable, para referenciarse a ella se debe de escribir como se declaro.

Las variables en OCCAM tienen asociado un tipo.

TIPO	DESCRIPCION
INT	Entero de igual tamaño al de la palabra del TRANSPUTER.
INT16	Entero de 16 bits
INT32	Entero de 32 bits
INT64	Entero de 64 bits
REAL32	Real de 32 bits
REAL64	Real de 64 bits
BYTE	Byte sin signo
BOOL	Booleano
TIMER	Timer (32 bits)

Figura 12. Tipos de OCCAM

Los enteros de cualquier longitud se consideran signados. La declaración de una variable se delimita con dos puntos, por ejemplo :

```
INT a,b :
INT32 c:
```

La asignación de una variable en OCCAM es denotada por := por ejemplo :

```
prop := 42
```

La declaración de constantes tiene la siguiente sintaxis:

```
VAL two IS 2.0 (REAL32)
```

En donde two contiene el valor de 2 y es de tipo REAL32, así en donde se encuentre la constante two su lugar es sustituido por el valor de dos.

III.4. Cadenas de caracteres[12]

Un caracter es declarado como del tipo BYTE:

'a' : Caracter ascii entre apóstrofes

Una cadena de caracteres es tratada como un arreglo de BYTES y delimitados por comillas:

"este es un string"

Los caracteres que no se deben de utilizar en cadenas son:

* ' "

Existen los caracteres de control como en el lenguaje C, representados por un asterisco que precede al caracter como una secuencia de escape:

*c	*C	Carriege return	=	'*#0D'
*n	*N	nueva línea	=	'*#0A'
*t	*T	tab	=	'*#08'
*s	*S	espacio	=	'*#20'
*'		apóstrofe		
*"		comillas		
**		asterisco		

Algunos BYTES se representan por *# seguido de dos dígitos hexadecimales:

BELL := '*#07'

Un comentario se escriben anteponiendo dos guiones.

-- este es un comentario

III.5. Ciclos[12]

Hay dos tipos de ciclos en OCCAM:

- El ciclo WHILE :

Sintaxis :

WHILE expresión que evalúa un booleano
... cuerpo del ciclo

La expresión de control para el ciclo es evaluada a un valor booleano TRUE o FALSE, el final de la construcción, WHILE depende de la indentación de las instrucciones que conformen su cuerpo.

- La replicación del proceso SEQ :

```
SEQ i= 0 FOR 100
  a := a + 1    -- proceso que se va a replicar
```

La variable i es implícitamente declarada con la replicación de SEQ, el salto es siempre de tamaño uno y el índice es de tipo entero.

Entiéndase como replicación, la creación de procesos similares, en este caso se crean procesos secuenciales similares.

III.6. Arreglos[12]

Un arreglo siempre inicia con un elemento cero. Los corchetes se utilizan para denotar un arreglo:

```
[10]INT un.arreglo:
```

En este se declara un arreglo de 10 enteros.

Los arreglos multidimensionales son posibles, y pueden ser considerados como arreglos de arreglos:

```
[3][7]INT coeficientes
```

Es un arreglo de 3 elementos, cada elemento es compuesto de 7 elementos individuales, un elemento individual puede ser accesado por:

```
a:=coeficientes[0][6]
```

Un subarreglo puede ser tratado como un solo elemento:

```
coeficientes[1]:=coeficientes[3]
```

III.7. Procedimientos[12]

Los procedimientos se definen conforme la siguiente sintaxis :

```
PROC nom.proc (type.1 variable1,variable2,type.3 variable3. . .)
    cualquier.cosa.de.lo.contrario
:
```

La palabra reservada PROC indica el inicio del procedimiento y el caracter : indica el final. Los parámetros se colocan entre paréntesis, cada uno con su tipo. El cuerpo del procedimiento es indentado por dos espacios e incluye declaraciones. Si el tipo de una variable es el mismo que el anterior, este tipo puede ser omitido y la lista de variables del mismo tipo quedarían separados por comas.

Los parámetros pueden ser pasados por valor o por referencia, para indicar que es por valor únicamente escriba la palabra reservada VAL antes del tipo de cada variables.

Y finalmente para llamar al procedimiento :

nom.proc (variable1.o.valor,variable2.o.valor. . .)

donde : nom.proc significa el nombre del procedimiento
variablen , n=1,2,3... significa las variables o valores que son enviados al procedimiento

III.8. Flujo de programación secuencial [12]

Los principales constructores para la manipulación de la secuencia de un programa son:

- Constructor IF

La construcción IF puede usarse para probar una ó varias condiciones.

Sintaxis :

```
IF
  a < 1
    . . . Acción uno
  b=0
    .  . Acción dos
  c > 2
       . Acción tres
  TRUE
    . . . Acción por default
```

Las pruebas se evalúan en el orden en que son escritas, en caso de que la constante TRUE sea omitida por el programador, y ninguna condición se cumple el TRANSPUTER pone en uno una bandera de error.

- Constructor CASE [12]

Mientras IF permite probar si cada condición es verdadera, CASE toma únicamente una expresión y la prueba contra las expresiones permisibles.

Sintaxis :

```
CASE expresión
  selector.1
    proceso.1
  selector.2
    proceso.2
  ELSE
    proceso.default
```

III.9. Comunicación y construcciones de procesos paralelos [12]

Ahora veremos algunos conceptos importantes de OCCAM, que van más involucrados al procesamiento en paralelo.

- Canales de comunicación

El procesamiento en paralelo no sería muy útil si los procesos no pudieran interactuar entre ellos, OCCAM utiliza el concepto de canales de comunicación para establecer la comunicación entre procesos.

En OCCAM la comunicación es:

SINCRONIZADA
SIN BUFFER
EN UN SOLO SENTIDO
CON TIPOS BIEN DEFINIDOS

Los canales se declaran de la siguiente forma:

Sintaxis :

CHAN OF TIPO Nombre

Donde: TIPO es un tipo de dato valido
Nombre es el nombre del canal.

P.ej:

```
CHAN OF INT canal.uno
CHAN OF BOOL habilita :
```

Los canales pueden ser utilizados para entrada o salida de datos.

Los canales de entrada únicamente pueden tener sitio dentro de una variable y el proceso ?, el cual representa la adquisición de datos desde un canal.

```
CHAN OF BOOL habilita :
BOOL ir :
SEQ
  habilita ? ir
```

Un canal de salida es similar, únicamente que ahora se utiliza el proceso !, sin embargo, una constante se puede enviar, al igual que alguna expresión.

```
CHAN OF BOOL habilita :
BOOL ir :
SEQ
  ir :=FALSE
  habilita ! ir
  habilita ! TRUE
```

- Constructor PAR [12]

El valor real de los canales es su uso en programas paralelos. Esto da un medio para que diferentes procesos puedan trabajar independientemente en una forma controlable y productiva y además cooperativa.
Si el proceso va a correr en paralelo, entonces la sentencia PAR es usada en lugar de SEQ. P.ej.

```
CHAN OF INT in, out, between
PAR
  WHILE TRUE
    INT x :
    SEQ
      in ? x
      between ! x
  WHILE TRUE
    INT x :
    SEQ
      between ? x
      out ! x
```

Estos procesos se representan así:

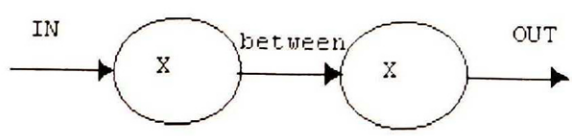


Figura 13. Inter - relación entre 2 procesos

Cuando una entrada aparece en IN, X recibe el valor y lo saca sobre between, simultáneamente el segundo proceso recibe la entrada desde between y la asigna a X, la cual contiene el valor y lo saca por el canal out.

- Replicación de PAR [12]

Cuando se replica el constructor PAR crea tantos procesos paralelos como lo indique el replicador. Hay una sola pieza de código pero cada proceso tiene su propia área de trabajo, la sintaxis es:

PAR i= 0 FOR N

- Constructor ALT

El constructor ALT es similar al case, pero más potente, esto es, CASE opera sobre una variable, ALT opera sobre canales, con esto permite que una entrada sea seleccionada desde el primer canal que esté listo de entre varios canales, y así se ejecute el proceso asociado con dicha entrada. El constructor ALT es el único modo para romper la comunicación sincronizada entre procesos. Esto permite tener un proceso con múltiples entradas y seleccionar alguna de ellas.

```
ALT
  chan.1 ? variable
    proceso.1
  chan.2 ? variable
    proceso.2
```

En términos de diagramas de procesos la construcción quedaría:

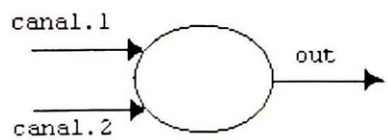


Figura 14 Canales y procesos

- Replicación de ALT[12]

El constructor ALT también puede ser replicado.

La sintaxis es:

ALT i= base FOR count

por ej.:

```
. . . declaraciones
[10]CHAN OF BYTE entrada :
CHAN OF BOOL alto :
SEQ
  IR:= TRUE
  WHILE IR
    ALT i:=0 FOR 10
      entrada[i] ? char
       out      ! char
```

- Protocolos de canales [12]

Un protocolo es una estructura definida por el usuario, la cual nos indica que tipos de datos se pueden transferir en un canal, por ej.

```
PROTOCOL PUNTO.TRES.D IS REAL32;REAL32;REAL32 :
```

El cual puede ser utilizado por:

```
CHAN OF PUNTO.TRES.D coordenada :
cordenada ?  x;y;z
```

Otro ej.

```
PROTOCOL KERMIT IS INT::[]BYTE;INT16 :
CHAN OF KERMIT from.computer to.computer :
SEQ
  from.computer  ?   len::string;checksum
```

Nótese que un Protocolo es una abstracción de los tipos de datos que puede transferir un canal.

III.10. Aritmética en OCCAM[12]

En OCCAM no existe precedencia de operadores, el programador emplea tantos paréntesis como requiera; y además cada operación válida únicamente corresponde a un operador con sus dos operandos (en caso de operadores binarios), los paréntesis únicamente asociarán operaciones que tengan esta estructura.

A continuación se presenta una tabla conteniendo todos los operadores validos en OCCAM.

OPERADORES ARITMETICOS	
+	adición
-	substracción
*	multiplicación
/	división
\	residuo
OPERADORES DE MODULO ARITMÉTICO	
PLUS	Módulo adición
MINUS	Módulo substracción
TIMES	Módulo multiplicación
OPERADORES DE BITS	
/\	Operación AND en BIT's
\/	Operación OR en BIT's
><	Operación OR EXCLUSIVO en BIT's
~	Operación NOT en BIT's
OPERADORES DE CORRIMIENTO	
>>	Corrimiento a la derecha
<<	Corrimiento a la izquierda
OPERADORES BOOLEANDOS	
AND	Operación AND en Booleanos
OR	Operación OR en Booleanos
NON	Operación NOT en Booleanos
OPERADORES COMPARACIÓN	
=	Operador IGUAL
<>	Operador NO IGUAL
<	Operador Menor que
>	Operador mayor que
<=	Operador menor o igual que
>=	Operador mayor igual que

Figura 15. Operadores

Dentro del conjunto de construcciones de OCCAM existen otros tipos los cuales son aplicables al desarrollo de otra clase de sistemas, por ejemplo los de tiempo real, para esto, el mismo procesador tiene un puerto de ocho bits con el que se comunica al exterior y así controle procesos de tiempo real.

Solamente se ha presentado un conjunto limitado de instrucciones del lenguaje de programación OCCAM , ya que son las que en mayor grado se utilizan en el desarrollo del preprocesador OBOCCAM.

- Constructor CASE [12]

Mientras IF permite probar si cada condición es verdadera, CASE toma únicamente una expresión y la prueba contra las expresiones permisibles.

Sintaxis :

```
CASE expresión
  selector.1
    proceso.1
  selector.2
    proceso.2
  ELSE
    proceso.default
```

III.9. Comunicación y construcciones de procesos paralelos [12]

Ahora veremos algunos conceptos importantes de OCCAM, que van más involucrados al procesamiento en paralelo.

- Canales de comunicación

El procesamiento en paralelo no sería muy útil si los procesos no pudieran interactuar entre ellos, OCCAM utiliza el concepto de canales de comunicación para establecer la comunicación entre procesos.

En OCCAM la comunicación es:

SINCRONIZADA
SIN BUFFER
EN UN SOLO SENTIDO
CON TIPOS BIEN DEFINIDOS

Los canales se declaran de la siguiente forma:

Sintaxis :

CHAN OF TIPO Nombre

Donde: TIPO es un tipo de dato valido
Nombre es el nombre del canal.

P.ej: CHAN OF INT canal.uno :
CHAN OF BOOL habilita :

Los canales pueden ser utilizados para entrada o salida de datos.

Los canales de entrada únicamente pueden tener sitio dentro de una variable y el proceso ?, el cual representa la adquisición de datos desde un canal.

```
CHAN OF BOOL habilita :
BOOL ir :
SEQ
  habilita ? ir
```

Un canal de salida es similar, únicamente que ahora se utiliza el proceso !, sin embargo, una constante se puede enviar, al igual que alguna expresión.

```
CHAN OF BOOL habilita :
BOOL ir :
SEQ
  ir :=FALSE
  habilita ! ir
  habilita ! TRUE
```

- Constructor PAR [12]

El valor real de los canales es su uso en programas paralelos. Esto da un medio para que diferentes procesos puedan trabajar independientemente en una forma controlable y productiva y además cooperativa.
Si el proceso va a correr en paralelo, entonces la sentencia PAR es usada en lugar de SEQ. P.ej.

```
CHAN OF INT in, out, between :
PAR
  WHILE TRUE
    INT x :
    SEQ
      in ? x
      between ! x
  WHILE TRUE
    INT x :
    SEQ
      between ? x
      out ! x
```

Estos procesos se representan así:

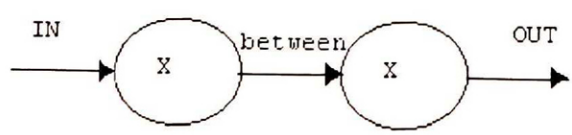


Figura 13. Inter - relación entre 2 procesos

Cuando una entrada aparece en IN, X recibe el valor y lo saca sobre between, simultáneamente el segundo proceso recibe la entrada desde between y la asigna a X, la cual contiene el valor y lo saca por el canal out.

- Replicación de PAR [12]

Cuando se replica el constructor PAR crea tantos procesos paralelos como lo indique el replicador. Hay una sola pieza de código pero cada proceso tiene su propia área de trabajo, la sintaxis es:

PAR i= 0 FOR N

- Constructor ALT

El constructor ALT es similar al case, pero más potente, esto es, CASE opera sobre una variable, ALT opera sobre canales, con esto permite que una entrada sea seleccionada desde el primer canal que esté listo de entre varios canales, y así se ejecute el proceso asociado con dicha entrada. El constructor ALT es el único modo para romper la comunicación sincronizada entre procesos. Esto permite tener un proceso con múltiples entradas y seleccionar alguna de ellas.

```
ALT
  chan.1 ? variable
    proceso.1
  chan.2 ? variable
    proceso.2
```

En términos de diagramas de procesos la construcción quedaría:

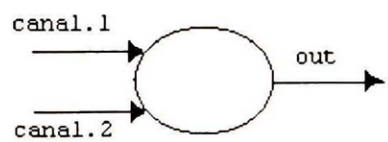


Figura 14. Canales y procesos

- Replicación de ALT[12]

El constructor ALT también puede ser replicado.

La sintaxis es:

ALT i= base FOR count

por ej.:

```
. . . declaraciones
[10]CHAN OF BYTE entrada :
CHAN OF BOOL alto :
SEQ
  IR:= TRUE
  WHILE IR
    ALT i:=0 FOR 10
      entrada[i] ? char
       out       ! char
```

- Protocolos de canales [12]

Un protocolo es una estructura definida por el usuario, la cual nos indica que tipos de datos se pueden transferir en un canal, por ej.

```
PROTOCOL PUNTO.TRES.D IS REAL32;REAL32;REAL32 :
```

El cual puede ser utilizado por:

```
CHAN OF PUNTO.TRES.D coordenada :
cordenada ?  x;y;z
```

Otro ej.

```
PROTOCOL KERMIT IS INT::[]BYTE;INT16 :
CHAN OF KERMIT from.computer to.computer :
SEQ
  from.computer  ?   len::string;checksum
```

Nótese que un Protocolo es una abstracción de los tipos de datos que puede transferir un canal.

III.10. Aritmética en OCCAM[12]

En OCCAM no existe precedencia de operadores, el programador emplea tantos paréntesis como requiera; y además cada operación válida únicamente corresponde a un operador con sus dos operandos (en caso de operadores binarios), los paréntesis únicamente asociarán operaciones que tengan esta estructura.

A continuación se presenta una tabla conteniendo todos los operadores validos en OCCAM.

OPERADORES ARITMÉTICOS	
+	adición
-	substracción
*	multiplicación
/	división
\	residuo
OPERADORES DE MODULO ARITMÉTICO	
PLUS	Módulo adición
MINUS	Módulo substracción
TIMES	Módulo multiplicación
OPERADORES DE BITS	
/\	Operación AND en BIT's
\/	Operación OR en BIT's
><	Operación OR EXCLUSIVO en BIT's
~	Operación NOT en BIT's
OPERADORES DE CORRIMIENTO	
>>	Corrimiento a la derecha
<<	Corrimiento a la izquierda
OPERADORES BOOLEANDOS	
AND	Operación AND en Booleanos
OR	Operación OR en Booleanos
NON	Operación NOT en Booleanos
OPERADORES COMPARACIÓN	
=	Operador IGUAL
<>	Operador NO IGUAL
<	Operador Menor que
>	Operador mayor que
<=	Operador menor o igual que
>=	Operador mayor igual que

Figura 15. Operadores

Dentro del conjunto de construcciones de OCCAM existen otros tipos los cuales son aplicables al desarrollo de otra clase de sistemas, por ejemplo los de tiempo real, para esto, el mismo procesador tiene un puerto de ocho bits con el que se comunica al exterior y así controle procesos de tiempo real.

Solamente se ha presentado un conjunto limitado de instrucciones del lenguaje de programación OCCAM , ya que son las que en mayor grado se utilizan en el desarrollo del preprocesador OBOCCAM.

III.11. Conclusiones

El lenguaje de programación OCCAM[6] se deriva del modelo matemático conocido como CSP implementado por Hoare, esencialmente OCCAM cuenta con mecanismos para la especificación de la concurrencia y primitivas de comunicación y sincronización por medio del paso de mensaje, además del manejo de construcciones para la comunicación selectiva y guardias de dijkstra que se emplean con operaciones de entrada y salida de mensajes.

En conjunto todos los atributos de programación que fueron implementados en OCCAM nos dan un lenguaje flexible y poderoso para la construcción de sistemas concurrentes y con ello llegar a explotar el procesamiento en paralelo.

CAPITULO IV

PROGRAMACIÓN DE OBJETOS EN OCCAM

IV.1. Introducción

Conjuntar el paradigma de objetos y programación concurrente resulta muy natural y el cambio viene dado en el momento en que utilizamos un objeto concurrente[6].

Hasta aquí se ha presentado, los objetivos del trabajo, la arquitectura de la TRANSPUTER y los mecanismos de programación de OCCAM, incluyendo los mecanismos de comunicación entre objetos, el cual fue presentado en el capitulo dos, en el presente capítulo se analiza la forma de conceptualizar los objetos desde el punto de vista de OCCAM.

Como se vio en el primer capítulo, un objeto es el encapsulamiento de datos y sus métodos; principalmente esta constituido de los siguientes métodos: constructor, destructor y algunos otros definidos por el usuario, los cuales en conjunto definen el comportamiento del objeto.

Aquí se va a presentar los esquemas de como se definen y construyen los objetos a partir de las sentencias que componen el lenguaje de programación OCCAM.

No debemos de olvidar que los objetos se ejecutan concurrentes y se comunican por medio de un conjunto de protocolos.

IV.2. Representación de un objeto

En OCCAM, la construcción que nos representa un objeto es el proceso, este tendrá el código correspondiente para administrar las diferentes llamadas a los miembros del objetos, es decir, los métodos y los datos; además identifica a que instancia del objeto se invoca en un determinado momento.

En otras palabras un proceso será quien en base a su código tenga la funcionalidad de un administrador de objetos y ofrezca todos los servicios necesarios a las instancias que sean generadas en base a la declaración que representa.

Las construcciones básicas de OCCAM para representar un objeto son:

- El proceso (Administra y se ejecuta concurrentemente)
- Canales y protocolos (Nos ayudan a comunicar los objetos)
- Procedimientos de OCCAM (Serán los métodos que utilizan los objetos)
- Arreglos de datos (Nos darían localidades de memoria para almacenar los datos de las instancias

Así, la representación de un objeto en OCCAM se muestra en la figura 16.

El proceso está asociado a procedimientos que nos representan los servicios del objeto por ejemplo: el de construcción ó el de destrucción, además, tiene espacio para almacenar los valores de las distintas instancias del objeto.

Además, el proceso por su naturaleza mantiene la posibilidad de ejecutarse concurrentemente y su código es construido en tiempo de compilación.

A este proceso se le llama administrador del objeto y se encuentra ligado al proceso de administración de comunicación, el cual le dará la interacción con otros administradores de objetos.

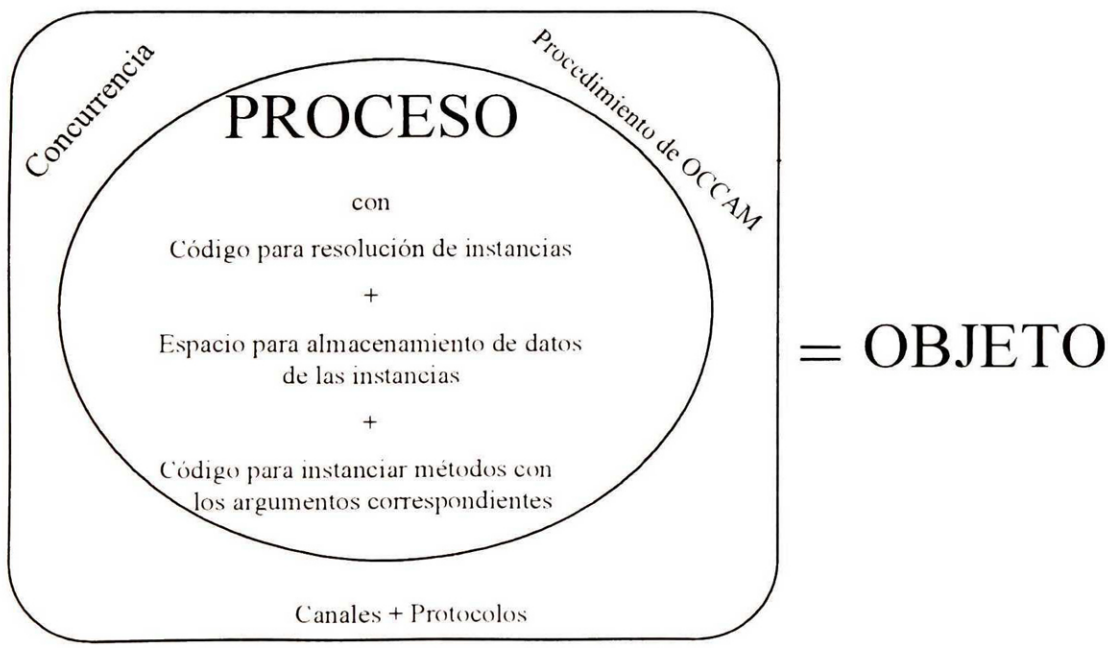


Figura 16. El Objeto

En general, el proceso mantendrá los mecanismos de atención a llamadas de otros objetos, cada vez que esto sucede el objeto invocará al procedimiento correspondiente, por ejemplo a los procedimientos de consulta y actualización de los datos de sus instancias y sobre todo el control necesario para la administración de su comportamiento.

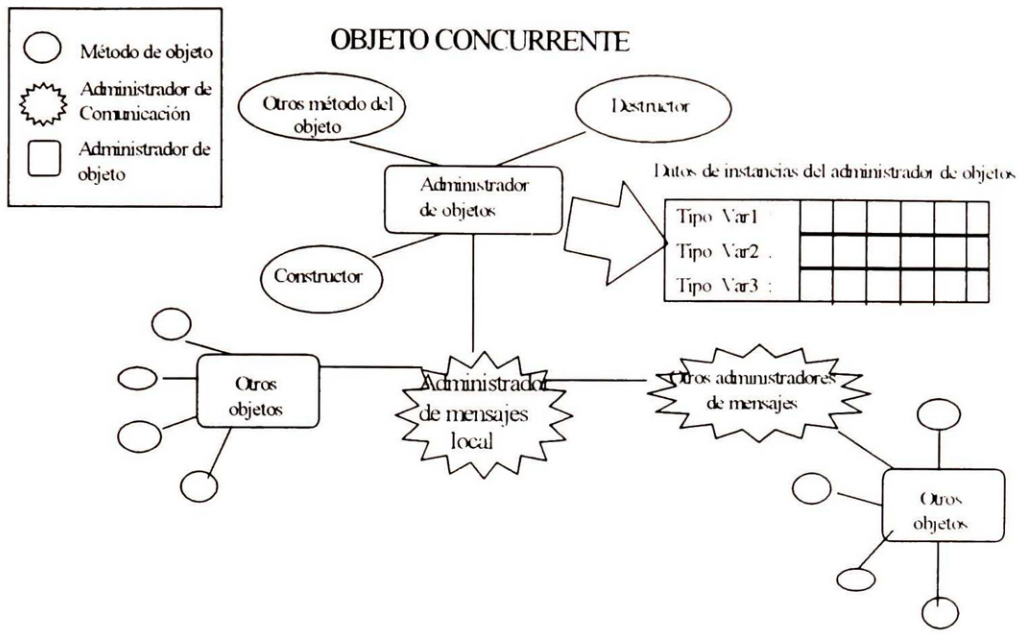


Figura 17. Inter - relación entre objetos concurrentes

En la figura 17 se muestra como se inter-relacionan los distintos objetos que se generan en tiempo de ejecución de un programa, estos objetos se ejecutan concurrentemente y los problemas de requerimientos de servicios es solucionada por el administrador de mensajes.

IV.3. Datos de un objeto

El tipo de datos más complejo de OCCAM es el arreglo, en nuestra implementación de OBOCCAM, se identifican básicamente dos tipos de objetos :

1).- Tipo de estructura de datos.
Objeto que tiene dos tipos de servicios, consulta y actualización.

2).- Objeto definidos por el usuario.
Un tipo de dato abstracto definido por el usuario, que nos representa un objeto

Analizaremos ahora como manipular y gestionar los datos de un objeto por medio del administrador de objetos que los representa.

Para mostrar la implementación de la gestión de datos lo haremos utilizando un objeto estructura de datos.

Una estructura se representa como el conjunto de datos almacenados.

Por ej.

```
STRUCT P IS
   INTEGER A;
   INTEGER B:
ENDSTR
```

Para representar al administrador de estructuras de datos, el cual administrará las instancias, utilizaremos un proceso de OCCAM el cual reserva memoria, por medio de un arreglo de tamaño finito, en donde, cada índice, indica la dirección de almacenamiento.
Por ej.

```
PROC OCCAMSTR (Canales de comunicación con el administrador de mensajes)
  [NUMMAX.DE.INSTANCIAS]INT A :
  [NUMMAX.DE.INSTANCIAS]INT B :
     . . .
    Código de administración de la estructura

: - - Fin del proceso administrador de la estructura
```

Por ejemplo, la localidad A[1] almacena el dato del entero A de la instancia identificada con 1.

Recordemos que las instancias son las diferentes variables del tipo de un objeto estructura de datos o de un objeto definido por el usuario.

En la selección del conjunto de protocolos asociado al objeto estructura, primero se establece cuales son sus funciones o cual es el propósito del objeto.

Es decir, al analizar el objeto estructura solo encontramos dos métodos, uno llamado "consulta" el cual su tarea es la de seleccionar el valor correspondiente a una instancia y regresarlo a la entidad que lo solicito (proceso o objeto), el otro método o función se llama "actualiza", el cual al recibir un valor determinado actualiza el correspondiente valor de dicha instancia.

Los protocolos de requerimiento de servicios, tendrán que indicar el método utilizado, por ejemplo consultar o actualizar, también debe indicar a que instancia pertenece, de que proceso es la requisición y la especificación del tipo de dato.

La instanciación es gestionada por el preprocesador, que, al encontrar un llamado a un método de un objeto, debe de colocar en su lugar, el mensaje que lo define completamente, este a su vez, es enviado a través del canal que está ligado directamente al administrador de mensajes y hace que llegue al destino deseado.

IV.4. Métodos de estructuras y de objetos

La administración de los métodos o servicios que ofrecen los objetos se implementa a través del proceso que es creado por OBOCCAM.

El proceso al encontrarse en comunicación con el administrador de mensajes, queda abierto a cualquier llamado de otro objeto.

Igualmente si alguna de sus instancias necesita alguno de sus métodos ésta puede invocarlo a través del administrador de mensajes.

En consecuencia cualquier invocación a un método del objeto, se hace por medio de un mensaje direccionado a su destino mediante el administrador de mensajes.

El llamado al método se realizaría en los siguientes casos:

- Para crear una instancia (Se llama a un método constructor, si existe)
- Para destruir una instancia, (Se llama a un método destructor, si existe)
- Desde un proceso o algún método de otro objeto que necesite el servicio
- Cuando se realiza un operación con un objeto (sobrecarga de operadores)

En cualquiera de los casos el preprocesador coloca en su lugar el código suficiente para que se logre el envió del mensaje que corresponda.

Al arribar el mensaje al proceso administrador de objetos, este resuelve dependiendo del tipo de mensaje, es decir, dentro del mensaje se encuentran los argumentos del servicio solicitado, por ej. si llega el requerimiento del cálculo de un factorial, se obtiene el número con el cual se van a realizar las operaciones, el administrador invoca al procedimiento que realiza la operación y regresa el resultado al objeto que hizo la requisición.

En la figura 18 se observa el algoritmo para la resolución de los distintos métodos que son invocados por otros objetos.

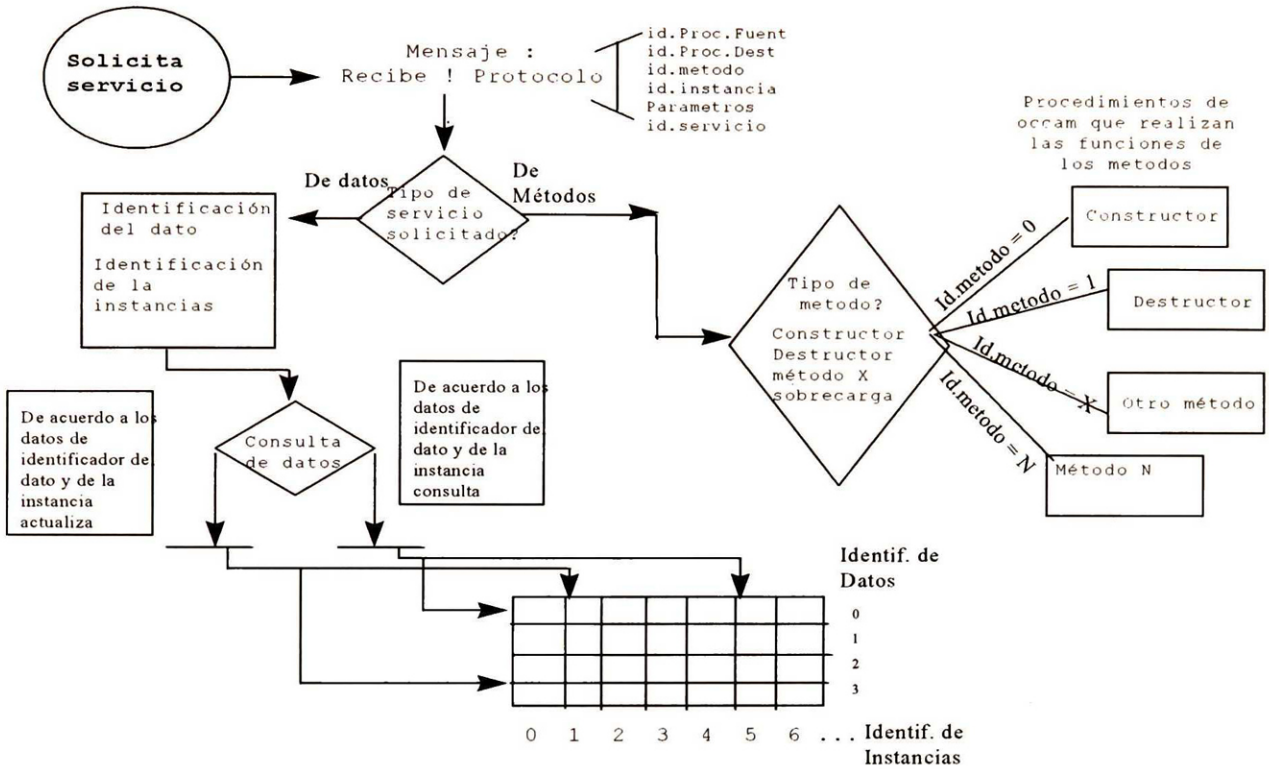


Figura 18. Resolución ó búsqueda de servicios de un objeto

IV.5. De métodos a procedimientos OCCAM.

En el inciso anterior se describió el algoritmo básico para la administración de objetos, a continuación se muestra el seudocódigo del proceso administrador.

```
PROC Objeto(CHAN OF INT to.AdmoMsg, from.AdmoMsg)
[MAXSIZEINSTANCIAS]INT dato.a, dato.b:
VARAUX INT :
SEQ
  WHILE TRUE
    ALT
      from.AdmoMsg ? CASE          -- VERIFICA ARRIBO DE MENSAJES
        A.BYTE;DATO.ID;AUXBY;STRNUM.INST;OPER;PROC.ID;PROC.D.ID
          -- Verifica que instancia es por medio de STRNUM.INST
          -- Verifica que tipo de operación se va realizar con el dato
          -- OPER indica si es consulta o actualización
          -- Retorna valores en caso de que sea una consulta

        A.MTHD;ID.DATO;N.PARAM::[]PARAM;Id.INTANCIA;Id.Oper;Proc; Dest
          -- Selecciona la instancia
          -- Selecciona el método
          -- obtiene el valor de las variables que vienen dentro de:
          -- []PARAM el número de parámetros viene en N.
          -- Llama al PROCEDIMIENTO
:
```

El proceso se ejecuta como un ciclo infinito, y escucha continuamente a través del canal que esta ligado al administrador de mensajes.

Cuando arriba un mensaje se verifican cada uno de los valores que vienen en él, estos valores son revisados en base a un protocolo ya definido Por ej.:

A.MTHD;ID.DATO;N.PARAM::[]PARAM;Id.INTANCIA;Id.Oper;Proc; Dest

A.MTHD - Indica que se trata de un llamado a un método.

ID.DATO .- Cuando se consulta un determinado tipo de dato, en el llamado a un método no se utiliza, pero ayuda a identificar a un determinado método en los casos en que se utiliza polimorfismo de funciones.

N.PARAM::[]PARAM :- Indica el número de parámetros y una cadena de caracteres en la que vienen concatenados los valores para cada uno, estos valores se separan por medio de un espacio.

Id.INTANCIA.- Indica la instancia que origino el llamado.

Id.Oper.- Identifica que método se va a ejecutar.

Proc.- Proceso fuente : Es el procesos que solicita los servicios

Dest.- Proceso destino : Es el proceso que ofrece el servicio.

En el código se va seleccionando las diferentes opciones que pueden dar como servicio, así cuando llega un mensaje con una etiqueta A.MTHD denota que se trata de una solicitud a un método, el método básicamente se trata de un procedimiento de OCCAM que es invocado como una sentencia normal de llamado a un procedimiento.

IV.6. Llamado desde un proceso administrador de objetos a un método

Los procedimientos representan a ios métodos, y es ejecutado desde el proceso administrador.

La información que se deben de tomar para la invocación del procedimiento son:

- Los valores de los parámetros del procedimiento
- La instancia que los invoca
- Los valores de los datos de las instancias en el caso de que su código las requiera.

Cada instancia se identifica con un índice, cuando la instancia llama a un método, el preprocesador la identifica dentro de su tabla de símbolos y llena en automático el valor correspondiente dentro del mensaje que será enviado al administrador del objeto.

Por ej.:

OBJETO.INSTANCIA_METODO(VAL_param, VAL_param2...)

El preprocesador genera código para convertir los valores de los parámetros a una cadena, la cual será incluida al mensaje.

También, para toda entidad(proceso o objeto) que contiene una instancia de algún objeto o realiza la requisición de un servicio, existe un canal especial que la liga al administrador de mensajes, el cual se utiliza para el envió y recibo de los mensajes hacia o desde un determinado administrador de objetos.

Así por medio del mensaje construido en base a un protocolo, solicita servicios a un administrador de objetos.

Por ejemplo la llamada anterior se sustituirá por el siguiente seudocódigo:

Código codificador de los parámetros
To.Canal.ADMIPROC ! A.MTHD;ID.INST;N.PARAM::]PARAM;Id.Dato;Id.Oper;Proc; Dest
From.Canal.ADMIPROC ? A.MTHD;ID.INST;N.PARAM::]PARAM;Id.Dato;Id.Oper;Proc; Dest
Código para tratamiento de recibo

Inicialmente por medio del codificador de los valores de los parámetros, estos se convierten en una cadena, también se recaba la información necesaria y la adiciona al mensaje para que sea enviado.

En este llamado se deduce que es una solicitud de un servicio de ejecución de un método, por la etiqueta A.MTHD

El envió se realiza por medio del proceso de salida ! que significa que se enviara el mensaje a través del canal llamado To.Canal.ADMIPROC

Se puede realiza en este caso un recibo de mensaje por medio del proceso **?**, esto es el caso de que se busque la sincronización entre procesos o se quiera recibir un valor de regreso.

Todo el código necesario para el tratamiento de parámetros y de la información que sea necesaria para el envió del mensaje es adicionado por el preprocesador y es invisible al desarrollador.

IV.7. Procedimiento de construcción y destrucción del objeto

Los procedimientos de construcción y destrucción son ejecutados cuando se crea una nueva instancia o cuando esta se destruye, respectivamente.

Estos códigos son adicionados por el programador y generalmente en la construcción se inicializan variables y se ejecutan algunos procesos que serán utilizados como recursos por el objeto.

Similarmente los códigos de terminación o destrucción de procesos, restablecen los recursos que utilizaron como memoria, canales de comunicación, servicios de entrada o salida, etc.

El llamado y el tratamiento de estos métodos, son similares a los métodos normales de los objetos y la única diferencia está en llamado.

IV.8. procedimientos de sobrecarga

Otros métodos que intervienen en la vida de un objeto son los procesos que realizan una tarea especial, en el momento que alguna de sus instancias aparece en una operación (aritmética, lógica etc.).

El tratamiento es similar, pero difiere en tiempo de compilación ya que creamos un nuevo procedimiento el cual utiliza los valores del objeto como parámetros.

Un problema grave es que OCCAM no brinda ayuda para la gestión dinámica de procesos y memoria por lo que la utilización de valores en tiempo de ejecución resulta difícil, es por eso que el acercamiento que se da es escaso y rudimentario.

Este acercamiento es similar al que se le da a un método común del objeto.

IV.9. Canales de comunicación

Los canales son de gran utilidad en el desarrollo de OBOCCAM, ya que por medio de ellos implementamos el paso de mensajes, esto se logra a través del administrador de mensajes, los mensajes tienen una estructura definida que es el conjunto de protocolos a los que responde un determinado objeto. En la siguiente sección detallamos estos protocolos.

Como se vio anteriormente los mensajes para los llamados y requisiciones de servicios son implementados por el preprocesador y son generalmente transparentes al usuario.

En general, se han establecido una serie de mecanismos para resolver algunos problemas presentados en la extensión, aunque no todos ellos fueron implementados en el preprocesador se trato de tenerlo lo más apegado a los mecanismos que se han presentado.

IV.10. Conclusiones

Las ideas básicas que componen el modelo de OBOCCAM son sencillas y fáciles de implementar, aunque resulta en ocasiones problemático, el encontrar soluciones optimas, debido a la ausencia de instrucciones para el manejo de memoria dinámica.

El preprocesamiento realizado por OBOCCAM transcribe todas aquellas declaraciones de objetos y las traduce a su equivalente en lenguaje OCCAM, se explico este mecanismo y se centro en la implementación del objeto y de los servicios que ofrece, para lograr esto se tuvieron que resolver los siguientes puntos los cuales se describen en los siguientes capítulos:

1. La comunicación para el llamado a objetos remotos.
2.- El administrador de mensajes.
3.- El administrador de un objeto definido por el usuario.
4.- El administrador de estructuras de datos definidas por el usuario.
5.- El llamado de métodos y datos (miembros del objeto).
6- La generación del código objeto para cada uno de los casos
7.- Las sintaxis convenientes para los distintos casos.

Se pueden hacer mejoras al modelo desarrollado para OBOCCAM, por ejemplo, optimización de sus algoritmos, implementar nuevos mecanismo de administradores de objetos, de comunicación, de llamado a miembros, etc. ya que se realizaron mecanismos básicos que con el paso del tiempo se pueden mejorar y seleccionar algunas otras soluciones que exploten el área de la programación concurrente orientada a objetos.

CAPITULO V

OBOCCAM: OCCAM BASADO EN OBJETOS

V.1. Introducción

En el presente capítulo profundizaremos en el proyecto y explicaremos las características esenciales de OBOCCAM, por ej. el modelo de programación, las palabras claves que se incorporan, su sintaxis, funcionamiento e implementación, el esquema básico de la comunicación entre objetos, etc. para que al final conozcamos la nueva funcionalidad que se pretendió con la extensión.

V.2. Programación en OBOCCAM

Independientemente de la programación concurrente que ofrece OCCAM aquí se explica la extensión, para llegar a OBOCCAM.

Los componentes básicos de OBOCCAM que se discuten a continuación son:

a) Estructura de datos en OBOCCAM
b) Componentes del Objeto en OBOCCAM
c) Método de comunicación simple entre objetos

V.2.1. La estructura de datos en OBOCCAM se compone de

- Datos
- Métodos (consulta y actualiza)

La estructura de datos es un objeto concurrente limitado en él cual ya se encuentra establecida su conformación como un conjunto de datos de diferentes tipo y dos métodos:

Consulta: que es llamado por algún objeto que necesite el servicio de conocer un valor de un dato de la estructura.

Actualiza: es aquel método del objeto estructura, el cual un proceso modifica el valor de una variable perteneciente a la estructura.

Por ej.

```
STRUCT  Ejemplo IS INT x, INT y :
. . .

1)      x1 := Ejemplo_x
2)      Ejemplo_y : = y2
```

Primero se define una estructura de datos llamada Ejemplo cuyas variables enteras son INT x, INT y, además los enunciados 1 y 2, se asigna el valor de la variable "Ejemplo_x" de la estructura a la variable "x1", mediante el llamado de

consulta y en el siguiente enunciado se asigna el valor de la variable "y2" al la variable "Ejemplo_y " de la estructura de datos.

Los métodos ***consulta*** y ***actualiza*** de la estructura quedan implícitos y transparentes para el programador.

V.2.2. Los objetos en OBOCCAM se componen de :

- Datos : Espacios de memoria en donde se almacenan los datos miembros del objeto
- Método constructor : Método cuya función es la de inicializar correctamente al objeto
- Método destructor : Método cuya función es la de destruir todos aquellos recursos utilizados por alguna instancia en particular del objeto
- Métodos de comportamiento : Métodos que nos representan los servicios del objeto.
- Métodos de sobrecarga de operadores : Métodos cuya función es la de realizar correctamente las operaciones aritméticas y lógicas en las que se vean involucradas las instancias del objeto.

La idea básica es obtenida del lenguaje de programación C++, en el cual un objeto es la encapsulación de los datos y sus métodos, cuando un objeto es declarado inmediatamente se establece el constructor, en OBOCCAM el constructor es un proceso inicializador de los datos del objeto así como de sus métodos.

Cuando OBOCCAM se encuentra una declaración de un tipo de dato abstracto que representa un objeto, este inmediatamente crea un administrador del objeto, el cual es un proceso concurrente que decide sobre los diferentes métodos del objeto así como de sus datos.

Este administrador se encarga de dar los servicios necesarios a los diferentes objetos y procesos que se ejecutan concurrentemente y que requieren de algún método o dato.

Igualmente cuando se encuentra una declaración de un tipo de objeto inmediatamente se le solicita al administrador de este objeto, la ejecución del método constructor.

Al finalizar la vida del objeto, este es destruido mediante la ejecución del método destructor lo cual es implícito y transparente al usuario.

La invocación desde un objeto a alguno de sus datos utilizamos la misma conceptualización de la estructura.

Para esta invocación utilizamos, como se explico anteriormente, una sintaxis similar al de la estructura de datos, pero para indicar que se trata de un método,

utilizamos la notación para el llamado a un procedimiento, esto es, después del nombre del método colocamos entre paréntesis los parámetros necesarios.

Por ej.:

```
IMPL OBJECT Ejemplo IS INT x, INT y
  METHOD Ejemplo::Metodo(INT c,INT h)
:
  .
x1 := Ejemplo_x              -- Llamado a un dato del objeto
Ejemplo_Metodo(12,x1)        -- Llamado a un método del objeto
```

Además de los métodos tradicionales indicamos otros que se activan cuando OBOCCAM encuentra una operación con alguno de las instancias del objeto, esto corresponde a la sobrecarga de operadores.

Para el caso de la sobrecarga de operadores, sí durante el preproceso de un programa de OBOCCAM, se llega a encontrar una operación con una instancia de algún objeto, implícitamente se determina la ejecución del llamado al método que realiza dicha operación como si fuese un método. esto naturalmente es invisible para el usuario y sigue la filosofía del llamado a un método del objeto

Por ej. Sea la declaración :

```
IMPL OBJECT Ejemplo IS INT x, INT y
  METHOD Ejemplo::OPERATOR+ (INT c)
:
  .
OBJECT Ejemplo a,b,c :

a :=  b + 10          -- Llamado a un operador sobrecargado.
```

La cual nos representa a un objeto, con un método para sobrecargar la operación de adición.

V.2.3. Un método de comunicación simple entre objetos

Se han definido las funciones involucradas con los objetos y las estructuras de datos de OBOCCAM, solo nos falta conocer una más, en la cual se pretende la abstracción en la resolución de búsqueda en una red de procesadores a un método, que ofrece como un servicio un objeto concurrente.

Para lograr esto utilizamos un objeto llamado administrador de mensajes el cual implícitamente dentro de un programa de OBOCCAM se encarga de hacer llegar un mensaje a un objeto servidor.

Simplemente cuando el programador necesita un servicio que ofrece un objeto remoto, este lo instancia como si fuese un objeto local, la tarea del preprocesador

es la, de definir los mecanismos necesarios para realizar el llamado a través del administrador de comunicación entre objetos, el cual queda invisible para el programador.

Una vez que llega el mensaje solicitando un servicio, el administrador de objeto llama al método involucrado.

La comunicación entre procesos administradores de objetos, procesos de usuario y administradores de estructura de datos, se hace a través del administrador de mensajes, para ello se implementaron un conjunto de protocolos de mensajes que contienen la información necesario para el requerimiento de un servicio.

Con lo anterior se dio a conocer los mecanismos fundamentales de programación que ofrece OBOCCAM, a continuación veremos detalladamente su sintaxis, no debemos olvidar que las especificaciones para el manejo de concurrencia ya vienen incluidos en el lenguaje de programación de OCCAM del cual se derivó el lenguaje OBOCCAM.

V.3. Extensión del lenguaje de OBOCCAM

A continuación se muestran las palabras reservas, que se involucran con la extensión para que sean soportados los conceptos de programación basada en objetos:

IS	**:**	**[ ]**	**STRUCT**
( )	**_**	**DELETE**	**OBJECT**
IMPL	**|**	**DESTRUCT**	**CONSTRUCT**
DEFN	**::**	**METHOD**	**OPERATOR**

Figura 19 Nuevas palabras clave

Algunas de ellas ya son utilizadas en las construcciones del lenguaje de programación OCCAM, aunque en nuestra extensión la reutilizamos con otro tipo de propiedades relacionadas a la extensión.

V.4. Construcciones del Objeto

V.4.1. a) Definición de estructuras

La notación utilizada nos representa los siguientes términos :

Negrita : Una palabra reservada.

"Identificador" Un identificador de alguna variable definido por el usuario.

Tipo : Algún tipo de OCCAM definido por el usuario.

Una estructura se define de la siguiente manera:

Sintaxis:

STRUCT "IDENTIFIER" **IS** TIPO "IDENTIFIER" ; TIPO "IDENTIFIER"....
:

Donde TIPO : INT, REAL32, REAL64, INT16, INT64, BOOL, Etc.

"IDENTIFIER" identificador utilizado por el usuario.

Semántica :

Se declara una estructura mediante las palabras reservadas **STRUCT**, **IS**, **:** . Para identificar sus diferentes variables se asocian a un TIPO de OCCAM seguido de la palabra que lo identifica, para construir un identificador seguimos las mismas normas de OCCAM.

Por Ej. :

A continuación tenemos tres definiciones de estructuras Punto, Circulo y Error:

STRUCT Punto IS INT X; INT Y :

STRUCT Circulo IS REAL32 r; INT x; INT y :

STRUCT Error IS [20]BYTE Msg; INT Cod; BOOL True :

Donde:

Un punto es una entidad de datos que almacenan dos valores enteros, los cuales nos representan las coordenadas en donde se encuentra situado,

Un circulo contiene la información necesario de un circulo geométrico, esto es un radio y dos enteros que nos representan las coordenadas del centro de la figura.

La estructura Error que contiene el espacio para almacenar un mensaje, un valor entero que nos representa un código definido para el tipo de error y finamente un booleano que nos representa el sentido del error (positivo o negativo).

Para la instanciación de la estructura utilizamos el nombre de la estructura(por ejemplo Punto) seguido por un underline (_) y el nombre que identifica a la variable o elemento que pertenece a la estructura(por ejemplo x ó y), por ej:

Punto_x := 12

Error_Msg := "Error en la salida del objeto"

x1 := 12 ;

V.4.2. Declaración de objetos

La declaración de un objetos se realiza en dos partes la primera considerada como la implementación del objeto, en donde, se declaran los métodos, sus principales funciones y datos, en esta parte se anexa la programación que lleva cada uno de los métodos, operadores sobrecargados, los constructores y los destructores.

Sintaxis :

```
IMPL OBJETO "IDENTIFIER" IS TIPO "IDENTIFIER" ; TIPO "IDENTIFIER";...

  CONSTRUCT:: "IDENTIFIER" ( Argumentos similar, a OCCAM)

      -- Implementación del constructor del objeto
      . . .
  :

  METHOD  "IDENTIFIER" :: Metodo1( Argumentos, similar a OCCAM)
      - - Implementación del metodo1 del objeto
        . . .
  :
        . . . -- Otros métodos
:
```

La segunda parte de la declaración del objeto corresponde a su definición, la cual nos indica el prototipo general del objeto y nos sirve para referenciarlo como un elemento de una librería de objetos, esto es una colección de objetos construidos con anterioridad y que los reutilizamos en distintos programas.

Sintaxis :

```
DEFN OBJETO "IDENTIFIER" IS TIPO "IDENTIFIER" ; TIPO "IDENTIFIER"....

  CONSTRUCT:: "IDENTIFIER" (... ) :     -- Prototipos del objeto

  METHOD  "IDENTIFIER"::Metodo1(...) : --  Prototipos del objeto

        - Otros prototipos del objeto
:
```

La división en dos partes de la declaración de un objeto tiene como finalidad:

1.- Separar la implementación por el programador del tipo de dato abstracto del objeto y de la instanciación del mismo desde otro programa, aprovechando así una de las características principales de la orientación a objetos correspondiente al encapsulamiento,

Esto evita tener que programar componentes concurrentes, facilitando así la utilización de los mismos únicamente con el prototipo para invocarlos.

2.- Mantener librerías útiles para cualquier desarrollo concurrente.

Inicialmente hemos explicado como se lleva a cabo la declaración de un objeto aun falta profundizar en cuatro conceptos importantes de la implementación del objeto:

i) La declaración de constructores

ii) La declaración de destructores

iii) La declaración de métodos

iv) La sobrecarga de operadores

Posteriormente explicaremos estos incisos.

V.4.3. Declaración de los constructores, destructores y métodos

En el cuerpo de la implementación del objeto se declaran los constructores

Sintaxis :

```
CONSTRUCT:: "IDENTIFIER" (Argumentos , similar a OCCAM )
  SEQ      -- Un constructor común de OCCAM

           -- Declaración de variables locales

           -- Inicialización de variables del objeto

    -- Llamado a otros objetos concurrentes que
    -- contribuyen a la inicialización del proceso.

    -- Otras operaciones.

:
```

Semántica :

"IDENTIFIER" Representa el nombre del objeto
(Argumentos) : Los parámetros que recibe el proceso, estos se declaran similarmente a los parámetros de los procesos de OCCAM.
SEQ : Nos representa un proceso común de OCCAM.

Nota : Para la construcción del cuerpo del método, seguimos las mismas reglas sintácticas y semánticas que se utilizan en la construcción de procesos de OCCAM, esto es, dentro de este métodos se pueden utilizar cualquiera de las construcciones definidas para OCCAM.

Similarmente la declaración de un destructor:

Sintaxis :

```
DESTRUCT :: "IDENTIFIER" (Argumentos , similar a OCCAM )
  SEQ    -- Un destructor común de OCCAM
    -- Declaración de variables locales
    -- Código de terminación de las variables del objeto
    -- Llamado a otros objetos concurrentes que contribuyen a la
    -- finalización del proceso.
    -- Otras operaciones de terminación.
:
```

Semántica :

"IDENTIFIER" Representa el nombre del objeto
(Argumentos) : Los parámetros que recibe el proceso, estos se declaran similarmente a los parámetros de los procesos de OCCAM.
SEQ : Nos representa un proceso común de OCCAM.

Nota : Para la construcción del cuerpo del método, seguimos las mismas reglas sintácticas y semánticas que se utilizan en la construcción de procesos de OCCAM, esto es, dentro de este métodos se pueden utilizar cualquiera de las construcciones definidas para OCCAM.

Un método de un objeto se define:

Sintaxis :

```
METHOD "IDENTIFIER1" :: "IDENTIFIER2" (Argumentos , similar a OCCAM )
  SEQ
          -- Declaración de variables locales
          -- Algún proceso referido a la necesidades del diseño
          -- El proceso lleva la sintaxis de OCCAM.
:
```

Semántica :

"IDENTIFIER1" Nombre del objeto.
"IDENTIFIER2" Nombre del método del objeto.
(Argumentos) : Los parámetros que recibe el proceso, estos se declaran similarmente a los parámetros de los procesos de OCCAM.
SEQ : Nos representa un proceso común de OCCAM.

Nota : Para la construcción del cuerpo del método, seguimos las mismas reglas que se utilizan en la construcción de procesos de OCCAM, esto es, dentro de este métodos se pueden utilizar cualquiera de las construcciones definidas para OCCAM.

V.4.4. Declaración de la sobrecarga

La sobrecarga de operadores para OBOCCAM siguen la filosofía de los métodos, esto es, similar al inciso anterior, una sobrecarga de operador es un método cuya referencia es el nombre del objeto y el operador involucrado.

La única diferencia que se observa es en la instanciación, ya que solamente se invoca al "método operador" cuando en una operación sobrecargada se ve involucrada una instancia de algún objeto que ya fue definida con anterioridad.

Sintaxis :

```
METHOD"IDENTIFIER"::OPERATOR"Simbolo"(Argumentos , similar a OCCAM )
  SEQ
      --  Declaración de variables locales
      --  El proceso lleva la sintaxis de OCCAM.
:
```

Semántica :

"IDENTIFIER" Nombre del objeto.
"Simbolo" Un símbolo que represente alguna operación que soporte la sobrecarga(+, -, *, /, etc.)
(Argumentos) : Los parámetros que recibe el proceso, estos se declaran similarmente a los parámetros de los procesos de OCCAM.
SEQ : Nos representa un proceso común de OCCAM.

Nota : Para la construcción del cuerpo del método, seguimos las mismas reglas que se utilizan en la construcción de procesos de OCCAM, esto es, dentro de este métodos se pueden utilizar cualquiera de las construcciones definidas para OCCAM.

Ejemplo :

```
METHOD  Ejemplo :: OPERATOR + (INT x)
  SEQ
    INT h  : -
    BOOL  P :
    SEQ
      Ejemplo_x: = Ejemplo_x + x

:
```

En el ejemplo, se muestra la sobrecarga del operador +, en el cual se indica como se realizará la operación, en este caso, solamente se recibe como parámetro el entero que se adicionara a los datos miembros del objeto, suponiendo que solamente existe uno.

V.5. Esquema del funcionamiento de un objeto y su implementación

En términos generales la implementación de un objeto se resume en :

```
IMPL OBJETO  "IDENTIFIER"  IS TIPO "IDENTIFIER" ; TIPO "IDENTIFIER"....
  CONSTRUCT:: "IDENTIFIER" (Argumentos , similar a OCCAM )
    SEQ              -- Un constructor común de OCCAM
      -- Declaración de variables locales
      -- Inicialización de variables del objeto
      -- Llamado a otros objetos concurrentes que contribuyen a la inicialización del proceso.
    .
    DESTRUCT :: "IDENTIFIER" (Argumentos , similar a OCCAM )
      SEQ            -- Un constructor común de OCCAM
        -- Declaración de variables locales
        -- Código de terminación de las variables del objeto
        -- Llamado a otros objetos concurrentes que contribuyen a la finalización
        -- del proceso.
      .
    METHOD  "IDENTIFIER1" :: "IDENTIFIER2"  (Argumentos , similar a OCCAM )
      SEQ
        -- Declaración de variables locales
        -- Algún proceso referido a la necesidades del diseño
        -- El proceso lleva la sintaxis de OCCAM.
    .
    METHOD "IDENTIFIER" :: OPERATOR "Simbolo" (Argumentos , similar a OCCAM )
      SEQ
        -- Declaración de variables locales
        -- Algún proceso referido a la necesidades del diseño
        -- El proceso lleva la sintaxis de OCCAM.
    .
:
```

La transcripción de construcciones de OBOCCAM a OCCAM resulta sencilla como se muestra en el siguiente ejemplo de un administrador de objetos[13] :

En el ejemplo, se explican cada una de las construcciones que componen a un administrador de objetos que representa a un objeto cuyo nombre es punto.

```
PROC punto(CHAN OF OBJENT ENTRA,SALE )
```

El administrador de objetos punto se construye como un proceso de OCCAM en el que se tienen dos canales, uno se utiliza para el arribo de mensajes mientras que el segundo se utiliza como salida de mensaje, ambos canales se conectan al administrador de mensajes.

Se describe ampliamente sobre los administradores de mensajes más adelante.

```
  [10]INT x :        -- VARIABLES DEL OBJETO
  [10]INT y :
  [10]BYTE i :
  [10]BYTE R :
```

[13] Recordemos que existe una transcripción del código de OBOCCAM a OCCAM en donde un objeto se convierte en un proceso administrador de un determinado tipo de objetos.

Las variables locales del objeto (miembros datos) se manipulan como arreglos en donde cada posición indica una instancia del objeto. Por lo tanto, cuando un objeto externo solicita un servicio con respecto a un dato miembro se incluye dentro del mensaje un indicador que representa la instancia a la pertenece.

Dentro del administrador del objeto también se definen otras variables que son utilizadas como auxiliares.

```
SEQ
  PROC.ID:=1
  WHILE TRUE
```

Se inicia el código mediante un constructor secuencial y se crea un ciclo infinito que indica que el proceso administrador de objetos estará ejecutándose indefinidamente ó hasta que no existan más instancias de este.

En el siguiente código se muestra una construcción ALT que espera el arribo de mensajes con protocolos predefinidos; el mensaje es el medio con el que otro objeto ó una instancia solicita un servicio.

La construcción ALT compara el mensaje de arribo con los distintos protocolos que acepta, cuando el mensaje es validado correctamente se ejecuta el código correspondiente.

En el primer tipo de protocolo se espera una solicitud de un dato de tipo BYTE en el cual se tienen dos identificadores de dato, sí observamos, en este caso, el administrador de objetos esta compuesto por cuatro datos, dos enteros y dos BYTE´S, por los que en esta sección se espera y se distingue cual de los dos byte, está solicitando el servicio, que a su vez puede ser de consulta o de actualización.

```
ALT
  ENTRA ? CASE          -- VERIFICA ARRIBO DE MENSAJES
    A.BYTE;DATO.ID;AUXBY;STRNUM;OPER;PROC.ID;PROC.D.ID
      IF
        DATO.ID=0
          IF
            OPER=0
              SEQ
                AUXBY:=i[STRNUM]
                SALE ! A.BYTE;DATO.ID;AUXBY;STRNUM;OPER;PROC.D.ID;PROC.ID
            OPER=1
              SEQ
                i[STRNUM]:=AUXBY
        DATO.ID=1
          IF
            OPER=0
              SEQ
                AUXBY:=R[STRNUM]
                SALE ! A.BYTE;DATO.ID;AUXBY;STRNUM;OPER;PROC.D.ID;PROC.ID
            OPER=1
              SEQ
                R[STRNUM]:=AUXBY
```

En el código se muestra como se actualiza o se consulta el arreglo correspondiente, este mecanismo se utiliza para cualquier tipo de dato del objeto o

estructura de datos, como se muestra en el caso de los datos de tipo entero:

```
A.INT;DATO.ID;AUXI;STRNUM;OPER;PROC.ID;PROC.D.ID
  IF
   DATO.ID=0
     IF
       OPER=0
         SEQ
          AUXI:=x[STRNUM]
          SALE ! A.INT;DATO.ID;AUXI;STRNUM;OPER;PROC.D.ID;PROC.ID
       OPER=1
         SEQ
          x[STRNUM]:=AUXI
   DATO.ID=1
     IF
       OPER=0
         SEQ
          AUXI:=y[STRNUM]
          SALE ! A.INT;DATO.ID;AUXI;STRNUM;OPER;PROC.D.ID;PROC.ID
       OPER=1
         SEQ
          y[STRNUM]:=AUXI
```

Ahora nos encargaremos del constructor, similarmente como en el caso de los datos del objeto, para el constructor se espera un mensaje etiquetado de tal forma que identifica al constructor del objeto.

```
A.CONT;DATO.ID;AUXI::PARAM;STRNUM;OPER;PROC.ID;PROC.D.ID
  SEQ
   INT cx :
   INT cy :
   INT S
   INT  C
   BOOL Error
   [25]BYTE CAD :
   SEQ
     IF
       AUXI= 0
         SEQ
          S,C:=0,0
          SEQ R = 0 FOR AUXI
            SEQ
              CAD[S]:=PARAM[R]
              IF
                PARAM[R]='*S'
                 SEQ
                   S:=0
                   IF
                     C=0
                       STRINGTOINT(Error,cx, CAD)
                     C=1
                       STRINGTOINT(Error,cy, CAD)
                   C:=C+1
              S:=S+1
    punto_Const(cx,cy)
```

Aquí el constructor es representado como un procedimiento de OCCAM, que para ser llamado, se tiene que conocer sus parámetros, en este caso antes de llamar al procedimiento `punto_Const(cx,cy)` se tiene que obtener el valor de los

parámetros y asignárselos a las variables relacionadas y previamente definidas, para esto se hace un procedimiento de conversión de valores.

Los parámetros vienen en la cadena llamada PARAM contenida en el protocolo del mensaje.

Este mismo procedimiento se utiliza para el destructor, los métodos y el método operador.

```
A.DEST;DATO.ID;AUXI::PARAM;STRNUM;OPER;PROC.ID;PROC.D.ID
  SEQ
    INT dx
    INT dy :
    INT  C :
    BOOL Error :
    [25]BYTE CAD :
    INT S :
    SEQ
      IF
        AUXI= 0
          SEQ
            S,C:=0,0
            SEQ R = 0 FOR AUXI
              SEQ
                CAD[S]:=PARAM[R]
                IF
                  PARAM[R]='*S'
                    SEQ
                      S:=0
                      IF
                        C=0
                          STRINGTOINT(Error,dx, CAD)
                        C=1
                          STRINGTOINT(Error,dy, CAD)
                      C:=C+1
                S:=S+1
      punto_Destr(dx,dy)

A.MTHD;DATO.ID;AUXI::PARAM;STRNUM;OPER;PROC.ID;PROC.D.ID
  SEQ
    IF
      DATO.ID=0
        SEQ
          puntolocaliza()
```

Si el método no tiene parámetros, no necesita conversión para la obtención de ellos, también, sí observamos, se identifican por medio de la variable DATO.ID a los diferentes procedimiento que puede tener un objeto.

```
DATO.ID=1
  SEQ
    INT ty :
    INT  C
    BOOL Error :
    [25]BYTE CAD
    INT S :
    SEQ
     IF
       AUXI= 0
         SEQ
```

```
                S,C:=0,0
                SEQ R = 0 FOR AUXI
                  SEQ
                    CAD[S]:=PARAM[R]
                    IF
                      PARAM[R]='*S'
                        SEQ
                          S:=0
                          IF
                            C=0
                              STRINGTOINT(Error,ty, CAD)
                          C:=C+1
                    S:=S+1
          puntotraslada(ty)
```

Este es el cuerpo básico del administrador de objetos, en el se muestra como a la llegada de un mensaje con un protocolo determinado, selecciona la alternativa que se requiera para dicho mensaje.

Existen básicamente cuatro casos de invocaciones de métodos para un objeto:

i) Constructor
Se realiza cuando se crea una nueva instancia del objeto.

ii) Destructor
Se realiza cada vez que se destruye una instancia del objeto.

iii) Consulta ó actualiza de datos
Cuando algún proceso u objeto requiere de alguna consulta o actualización

iv) Métodos (Método u operador)
Se realiza cuando un procesos invoca a un método que ofrece un objeto o algún proceso que tenga alguna instancia de este objeto.

V.6. Invocación de los métodos y datos de un objeto

la instanciación de un dato, se realiza mediante el siguiente código:

a) De una estructura : Estructura_dato

b) De objeto : Objeto_dato

El llamado a un método se realiza mediante:

Sintaxis:

Objeto_Metodo(argumentos,...)

El caracter "_" especifica la combinación del objeto con el miembro del objeto

Sintaxis :

Objeto_Miembro

Semántica :

Objeto = La declaración de un objeto o una estructura de datos.
Miembro = Dato de la estructura de datos ó del objeto, ó un método del objeto con sus respectivos argumentos.

El llamado se hace de forma similar a un procedimiento

El preprocesador OBCCAM mediante el análisis detecta cuando se trata de una consulta o una actualización.

A nivel de OCCAM, la implementación de una invocación a un método, se realiza mediante el siguiente código :

```
CANAL ! A.MTHD; ID.PROC; N.PARAM::"21 23" ; STRNUM; OPER ; PROC.ID; PROC.D.ID
```

Donde :

CANAL :El nombre del canal de salida, el cual es colocado por OBOCCAM para qúe el proceso realice el llamado hacia un administrador del objeto o estructura y se comunique con ellos a través del administrador de mensajes.

A.MTHD : Solicitud de un servicio de un objeto.

ID.PROC : Identificador del método del objeto, al cual se le solicita.

N.PARAM : :"21 23" : Número de parámetros y una cadena con los valores de los parámetros separados por un espacio.

STRNUM : Instancia a la que se refiere el servicio de una estructura u objetos.

OPER : Parámetro no utilizado para los objetos o estructuras, pero para posibles extensiones, nos ayuda a resolver cuando un determinado método tiene diferentes facetas que dependen del llamado que se realice (polimorfismo).

PROC.ID : Proceso fuente

ROC.D.ID : Proceso destino.

El envió y recibo de mensajes se hace a través del administrador de mensajes.

V.7. La comunicación entre objetos

El mapeo de procesos a procesadores en una red de TRANSPUTER se conoce como configuración.

En esta los procesos se configuran para que se ejecuten sobre un TRANSPUTER en particular.

La comunicación entre TRANSPUTER son definidas en la descripción de la configuración.

Nosotros utilizamos una configuración adicional en la cual señalamos los objetos que se conectan a un determinado administrador de mensajes; así podemos tener N objetos conectados a M administradores de mensajes, los administradores de mensajes se interconectan entre sí, para permitir la comunicación.

Los administradores de mensajes reconocen a los objetos que se conectan a él, cada administrador tienen una tabla de ruteo de la red de objetos, al igual que la ruta para otros objetos que se ejecutan remotamente y que por medio de otro administrador de mensajes puede establecerse la comunicación.

El llamado a Objetos remotos se resuelve a través del administrador de mensajes

Para establecer el direccionamiento entre procesos, el administrador de mensajes maneja una tabla local de direccionamiento. Este mecanismo resulta bastante útil ya que es similar a la forma de como TCP/IP resuelve la resolución del nombramiento de direcciones IP

En OBOCCAM resulta más sencilla su implementación, observemos la figura 20.

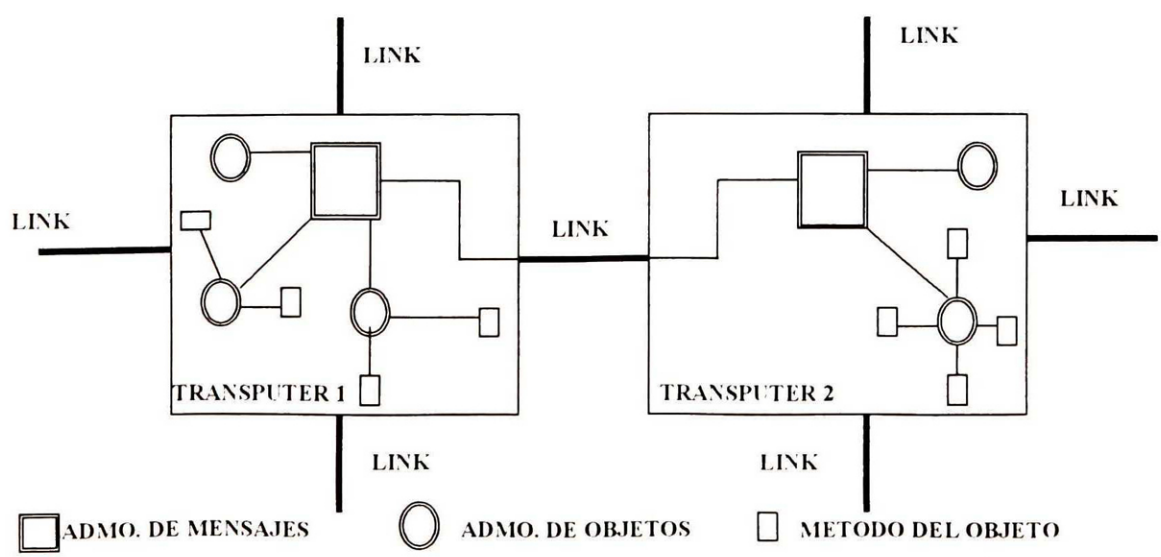


Figura 20. invocación a objetos remotos

En figura 20, tenemos varios objetos administradores de mensajes, a su vez estos se interconectan entre sí y mantienen canales directos con otros objetos (definidos por el usuario)

Los objetos definidos por el usuario se interconectan a través de un canal en el cual pasan los mensajes con distintos protocolos ya establecidos.

Cuando un objeto necesita el servicio de un método que ofrece otro objeto, este envía la petición como si fuera un objeto local :

Objeto_Metodo()

OBOCCAM al estar revisando el programa fuente y al encontrar el llamado al método, en su lugar coloca el mensaje correspondiente que se envía al administrador de mensajes el cual se encargará de hacerlo llegar hasta el objeto correspondiente.

El mecanismo de ruteo se realiza mediante los siguientes pasos

i) Arriba el mensaje al administrador de mensajes
ii) El administrador de mensajes lo recibe y verifica si es directo o indirecto.
iii) Si es un mensaje directo el administrador lo envía a través de canal que corresponda de acuerdo a la tabla de direccionamiento.
iv) por el contrario, si no es directo lo envía al administrador de mensajes correspondiente.

Para definir las ligas involucrados con los objetos .y a su vez la tabla de direccionamiento, el programador deberá de programarla, como si fuese una configuración de procesos a un determinado TRANSPUTER, el siguientes código nos muestra un aspecto de esta configuración.

```
tablaruta[0][0],tablaruta[0][1],tablaruta[0][2],tablaruta[0][3]:=0,1,2,3
tablaruta[1][0],tablaruta[1][1],tablaruta[1][2],tablaruta[1][3]:=0,0,0,0
tablaruta[2][0],tablaruta[2][1],tablaruta[2][2],tablaruta[2][3]:=0,1,2,3
liga[0],liga[1],liga[2],liga[3] := 0,1,2,3
```

El arreglo *tablaruta* tendrá el número del canal correspondiente por el cual fluirá el mensaje hacia el administrador de objeto correspondiente.

La información del arreglo tablaruta quedará representada de la siguiente forma:

Índice	0	1	2
	Hacia Objeto	Directo	liga
0	0	0	0
1	1	0	1
2	2	0	2
3	3	0	3

Figura 21. Tabla de ruteo entre objetos (ejemplo)

La primera columna indica el objeto con quien se puede comunicar el administrador de mensajes.

La segunda columna indica si la comunicación es directa (0) ó indirecta (1).

La tercera columna indica la liga de salida, idealmente el administrador de mensajes tiene ilimitado el numero de ligas con las que se pueden conectar los administradores de objetos.

El arreglo liga nos ayuda a encontrar el identificador del proceso que se encuentra conectado a una determinada liga

Índice	0	1	2	3
	0	1	2	...

Figura 22. Liga entre ejemplos (ejemplo)

Así el proceso cero se conecta a la liga 0.

El figura 23, muestra la configuración descrita en la figura 22.

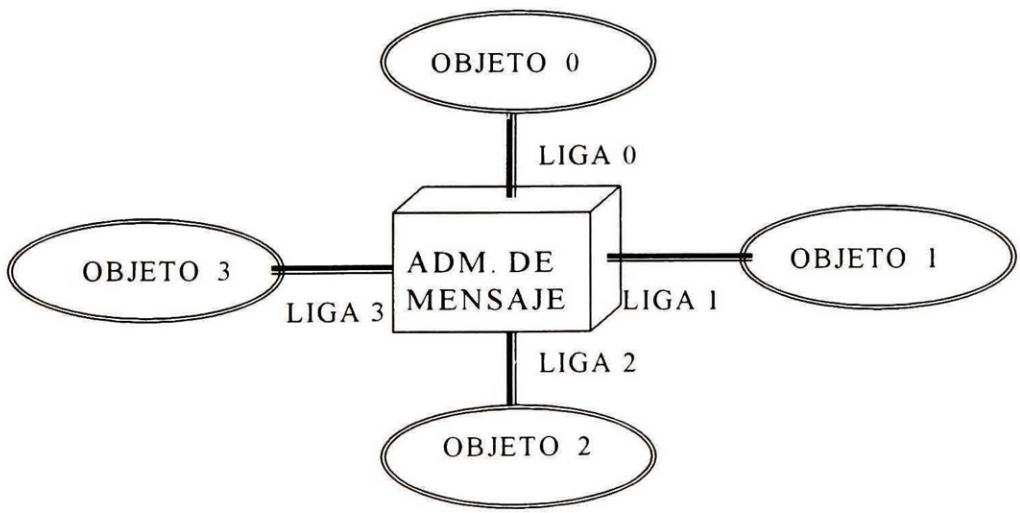


Figura 23. Ligas a objetos y el administrador de mensajes

La ligas son los canales de comunicación que se asignan entre un administrador de objetos y un administrador de mensajes, el administrador se encuentra "ligado" directamente con todos los objetos que caen en su dominio, así el administrador de mensajes tendrá en su tabla de direcciones a estos objetos marcados con una conexión directa.

Al final, la ejecución de los distintos administradores será en paralelo con otros procesos que haya colocado el programador.

```
PAR
  IMPRI(SALE[0],ENTRA[0])        --  PROCESO CERO
  STRNAME(SALE[1],ENTRA[1])      --  PROCESO UNO
  Punto(SALE[2],ENTRA[2])        --  PROCESO DOS
  Punto2(SALE[3],ENTRA[3])       --  PROCESO TRES
  ROUTER(ENTRA,SALE)             --  ADMINISTRADOR DE MENSAJES
```

V.8. Administrador de mensajes.

La parte fundamental para la comunicación entre objetos es llevada a cabo por el administrador de mensajes.

A continuación se explica el proceso conocido como administrador de mensajes.

```
PROC ROUTER ([NPROC]CHAN OF OBJENT ENTRA, SALE)
```

Como parámetros solamente utilizamos los canales que están conectados a los diferentes administradores de objetos.

```
WHILE TRUE
  SEQ
    diraux := -1
```

Se crea un ciclo infinito para detectar los arribos de mensajes provenientes de los distintos administradores de objetos.

```
ALT i=0 FOR NPROC
  ENTRA[i] ? CASE
    A.INT ; DATO.ID ; AUXI ; STRNUM ; OPER ; PROC.ID; PROC.D.ID
```

Por medio de la alternativa se verifica el arribo en los diferentes canales conectados a cada uno de los administradores de objetos.

```
IF r=0 FOR NMAXLIG
  tablaruta[0][r]=PROC.D.ID
    diraux:=r
```

En el IF anterior se localiza en tabla de direcciones los datos concernientes al objeto al cual se le va enviar el mensaje.

```
IF
  diraux = ( - 1 ) -- Existe un error o no se encuentra en la tabla.
    SALE [0] ! A.INT ; DATO.ID ; ERROR ; STRNUM ; OPER ; PROC.ID; PROC.D.ID
  TRUE
    IF
      tablaruta[1][diraux] =  0   -- liga directa
        SEQ
          SALE [liga[PROC.D.ID]] ! A.INT , DATO.ID ; AUXI ; STRNUM ; OPER , PROC.ID; PROC.D.ID
      TRUE
        |SALE [tablaruta[2][diraux]] ! A.INT ; DATO.ID , AUXI ; STRNUM ; OPER ; PROC.ID;
PROC.D.ID
```

En el primer IF se detecta si se encontró una dirección en la tabla de ruteo, si no se encontró se envía que existe un error al proceso 0, en este caso el proceso cero es utilizado para salida de vídeo.

En caso de que sea verdadero se selecciona la dirección valida, ya sea, si es una liga directa o indirecta.

Este algoritmo es utilizado para cada uno de los diferentes protocolos predefinidos, en este caso, se presentó para un tipo entero pero la misma idea se utiliza para todos los tipos de datos y métodos del objeto.

V.9. Protocolo utilizado y constantes utilizadas

Lo valores constantes son los siguientes :

```
VAL INT MAXDEFEST IS (INT 30)      -- N MAX DE ESTRUCTURAS
VAL INT NPROC IS (INT 30)          -- N MAX DE PROCESOS QUE UTILIZAN A LAS ESTRUC
VAL INT NDATO IS (INT 20)          -- N MAX DE DATOS DENTRO DE LA ESTRUCTURA
VAL INT NDEF IS (INT 20)           -- N MAX DE DEFINICIONES DEL TIPO ESTRUCTURA
VAL INT NTIPOSIM IS (INT 10)       -- N DE TIPOS SIM DENTRO DE LA ESTRUCTURA

PROTOCOL OBJENT        -- PROTOCOLO DE ENTRADA A UN PROCESO ESTRUCTURA
CASE
  A.INT  ; INT ; INT ; INT ; INT ; INT ;INT
  A.REAL ; INT ; REAL32; INT ; INT ; INT ; INT
  A.BOOL ; INT ; BOOL; INT ; INT ; INT ;INT
  A.BYTE ; INT ; BYTE; INT ; INT , INT , INT
  A.TIMER ; INT ; TIMER; INT ; INT ; INT ; INT
  A.INT16 ; INT ; INT16; INT ; INT ; INT ; INT
  A.INT32 ; INT ; INT32; INT ; INT ; INT ; INT
  A.INT64 ; INT ; INT64; INT ; INT ; INT ; INT
  A.REAL64 ; INT ; REAL64; INT , INT ; INT ; INT
  A.MTHD ; INT ; INT::[]BYTE; INT ; INT , INT ; INT
  A.OPER ; INT ; INT::[]BYTE; INT ; INT ; INT ; INT
  A.OUT ; INT ; INT::[]BYTE; INT ; INT ; INT ; INT
  A.INP ; INT ; INT::[]BYTE; INT ; INT ; INT ; INT
```

Cada protocolo nos va indicar un tipo de mensaje diferente, a continuación veremos una breve explicación del significado de cada protocolo.

El primer elemento A.XXXX Indica de que protocolo se trata, por ej., entero, real, booleano, método, etc.

El segundo elemento que es un entero, identifica una instancia del tipo de protocolo, es decir, si tenemos varios enteros dentro de un objeto, con este elemento podemos especificar a cual entero nos referenciamos.

El tercer elemento indica el valor del dato en especifico, si el protocolo transporta un entero, en el tercer elemento viene el valor real del entero. En el caso

del llamado de un método aquí viene el número de parámetros más la cadena de valores que sustituyen a los parámetros separados por un espacio.

El cuarto elemento que es de tipo entero, viene el número de instancia del objeto, ó estructura a la cual se le solicita un servicio. Por ej. si un administrador de objetos tiene cuatro instancias del mismo, con este se identifica a cual de ellas se esta refiriendo.

El quinto elemento, nos indica que tipo de operación se va realizar con algún dato, ya sea de consulta o actualización, para algún método con este elemento se puede controlar los métodos polimorficos.

El sexto elemento nos indica el identificador del procesos fuente.

El séptimo elemento nos indica el identificador del procesos destino.

V.10. Conclusiones

La sintaxis propuesta es completamente compatible con la utilizada por OCCAM, cubriendo así la extensión de OCCAM para la manipulación de objetos y una extensión compatible. La implementación de los administradores de objetos y de las estructuras de datos, son construidos por OBOCCAM en base a la extensión de la gramática de OCCAM, el administrador de mensajes es una librería de OCCAM que es incorporada una vez que OBOCCAM obtuvo el programa objeto, todo esto con el fin de integrar todas las herramientas que forman parte del trabajo y generar aplicaciones de objetos concurrentes.

Se presentaron la ideas básicas de los conceptos utilizados en la extensión, así como la sintaxis para su utilización, también se explicó el código generado en cada uno de los nuevos tópicos que son parte de OBOCCAM, con esto hemos logrado penetrar al área de la programación concurrente combinándola con el paradigma de objetos.

En este capitulo se explicaron puntos esenciales de la extensión, aunque en una primera etapa, que con el paso del tiempo se puede ir evolucionando y llegar a tener una herramienta poderosa para el desarrollo de aplicaciones concurrentes orientadas a objetos.

CAPITULO VI

APLICACIONES CON OBOCCAM

VI.1. Introducción

En el presente capítulo se explica el entorno en el que se desarrollo OBOCCAM, los pasos con los que construyó, las herramientas utilizadas, las técnicas de programación que se siguieron, entre otras cosas.

También se ejemplifica la utilización de varias construcciones del lenguaje OBOCCAM por medio de ejemplos, como son la creación de objetos, de estructuras la requisición de servicios, etc.

VI.2. Entorno de desarrollo

La plataforma de desarrollo se dividió en dos partes:

- Hardware : Los componentes electrónicos que utilizados para test.

  a) La microcomputadora (PC) que se utiliza para la entrada y salida de datos, y la carga del programa a la TRANSPUTER; sus características son las siguientes:
     1) un procesador Intel 486.
     2) 4MB de Ram.
     3) un disco duro de 80Mb.
     4) Floppy de 3.5 y 5.25

  b) El MONOPUTER que es una tarjeta con un solo TRANSPUTER cuenta con las siguientes características:
     1) Un procesador de 32 bits del tipo T805
     2) 2Mb de memoria privada
     3) 4 enlaces de comunicación a 100Mbits/seg
     4) Coprocesador de punto flotante

- Software: Programas con los que se desarrollo y se realizaron las pruebas.

  a) El sistemas operativo de la PC D.O.S V, 6.5
  b) El Compilador OCCAM 2
  c) El lenguaje C de Borlandc V 3.0
  d) PC YACC Se utilizo para generar el analizador sintáctico
  e) PC LEX Se utilizo para generar el analizador léxico

El diseño del modelo conceptual de la programación de OBOCCAM, se realizó basándose en los conceptos de objetos que utiliza C++, llevándolos al ámbito concurrente de OCCAM. Las razones por las que se selecciono como base C++ se

resumen en :

- Es un lenguaje que soporta los principales conceptos de la programación orientada a objetos (Clases, Tipo de datos abstractos, polimorfismo, sobrecarga, etc.).

- En la sección de computación del CINVESTAV se desarrollo un lenguaje concurrente derivado de C++ [8] con lo que se identifico como un excelente esquema para la implementación de objetos concurrentes.

- Ofrece técnicas de programación orientada a objetos que muchos otros lenguajes no las soportan como la abstracción de dividir una clase en definición e implementación.

En la construcción del preprocesador OBOCCAM se utilizaron las técnicas de desarrollo de compiladores en base a los siguientes pasos :

a) Se incluyeron las palabras clave de OCCAM y se le agregaron las nuevas para soportar la extensión .

b) En base al conjunto de palabras clave se diseño e implemento el analizador léxico.

c) Se tomo la gramática en Backus Noum Form de OCCAM como valido y se le anexaron las nuevas producciones que comprenden a la extensión.

d) El análisis de las nuevas producciones se utilizaron algunas que OCCAM tiene implementadas y se diseñaron otras que comprenden a la extensión.

e) En base a esta gramática extendida se construyo el analizador sintáctico.

d) Se desarrollaron los módulos correspondientes al preprocesador de OBOCCAM como son manejo de errores, reglas de traducción de código etc. y se iniciaron las pruebas del nuevo entorno.

VI.3. Utilizando a OBOCCAM

En el ejemplo que se explica a continuación se muestra la utilización de :

- Librerías adicionales para el manejo de objetos.
- Declaración de una estructura.
- Utilización de los datos de una estructura.
- Declaración de objetos
- Llamado a los métodos de un objeto
- Utilización de objetos para el manejo de entrada y salida
- El esquema de comunicación
- Llamados desde procesos
- El papel del administrador de mensajes.

Cada uno de los puntos anteriores se explican detalladamente a continuación:

VI.4. Librerías adicionales para el manejo de objetos

```
#INCLUDE "OBJETO.INC"
#INCLUDE "hostio.inc"
PROC COMUNICA(CHAN OF SP fs, ts, []INT memory)
  #USE "HOSTIO.LIB"
  #USE "CONVERT.LIB"
  #USE "OBJETO.LIB"
```

En el párrafo anterior se muestra dos librerías que son utilizadas por algunas de las palabras reservadas que fueron implementadas en la extensión, Por ej. el archivo "OBJETO.INC" contiene todas las constantes como son el número máximo de procesos, el número máximo de ligas por administrador de mensajes etc. La librería "OBJETO.LIB" contiene funciones que son utilizadas por los procesos que construye automáticamente OBOCCAM como son librerías de conversión, etc.

VI.5. Declaración de una estructura.

Observemos la siguiente declaración :

```
STRUCT STRNAME IS INT R1 ; INT R2:
```

En el ejemplo se utiliza una estructura de datos STRNAME, la cual atiende solicitudes de dos objetos llamados OBJECTO UNO y OBJECTO DOS, que se explican más adelante, estos objetos consultan y modifican las variables de la estructura de datos concurrentemente.

VI.6. Utilización de los datos de una estructura.

Un objeto que consulta al contenido de una variable miembro de una estructura, lo realiza mediante la siguiente instrucción :

```
X:=STRNAME_R1
```

En cambio para la actualización o modificación del contenido se especifica como :

```
STRNAME_R1:=X
```

En ambos casos se sigue la sintaxis definida en el capitulo anterior haciendo la referencia al nombre de la estructura y de la variable miembro separados por el caracter "_".

VI.7. Declaración de objetos .

```
IMPL OBJECT UNO IS INT IDEN
    METHOD UNO::RECIBE()
      INT X :
      SEQ
        SEQ
          PROC.ID,PROC.D.ID:=2,1
          DATO.ID,STRNUM,X:=0,0,13
          STRNAME_R1:=X
          X:=STRNAME_R1
          PROC.ID,PROC.D.ID:=2,0
          OPER:=0
    IMPRIME_ENTERO(X)
     :
  :
```

Aquí se ejemplifica la declaración de un objeto con un solo método llamado **METHOD UNO::RECIBE()** en donde su unica función es la de cambiar el valor de un dato miembro de la estructura y posteriormente consultarlo para que al final requiera el servicio `ENTERO(X)` del objeto `IMPRIME` , este objetos se ejecuta concurrentemente con otro proceso que realiza la misma función, en donde, ambos accesan a la misma estructura.

VI.8. Instanciación a los métodos de un objeto

Similarmente se realiza el llamado a un método de un objeto, por ej. en código anterior se utiliza el llamado al método Entero del objeto imprime.

```
IMPRIME_ENTERO(X)
```

VI.9. Utilización de objetos para el manejo de entrada y salida

```
IMPL OBJECT IMPRIME IS INT IDEN
    METHOD IMPRIME::ENTERO(INT X)
      SEQ
        PROC.ID,PROC.D.ID:=0,1
        DATO.ID,STRNUM:=0,0
        so.write.int(fs,ts,X,10)
      :
```

La función de este objeto es enviar un dato a la salida de vídeo, en este caso, como el acceso lo controla el administrador de mensajes, se puede compartir entre dos objetos el recurso de salida de video.

Aunque no es un mecanismo de OCCAM el compartir recursos, esta es una buena forma de implementarlo,

VI.10. El esquema de comunicación

La comunicación entre los distintos objetos es transparente para el usuario y esta implícito por OBOCCAM.

VI.11. Invocación desde objetos

```
PROC EJECUTOR()
    SEQ
      WHILE TRUE
        SEQ
          OPER,PROC.D.ID:=0,2
          PROC.ID,DATO.ID,STRNUM:=4,0,0
          UNO_RECIBE()
          OPER,PROC.D.ID:=0,3
          PROC.ID,DATO.ID,STRNUM:=4,0,0
          DOS_RECIBE()
  :
```

En la programación con OCCAM, se utiliza el concepto de proceso para definir a una entidad que se ejecuta concurrentemente, en este caso se ejemplifica como un procesos utiliza los servicios que ofrecen los objetos concurrentes.

VI.12. El administrador de mensajes.

La figura 24 nos muestra la interacción entre los administradores de objetos generados por OBOCCAM representados por medio de un circulo y el administrador de mensajes representado por un recuadro.

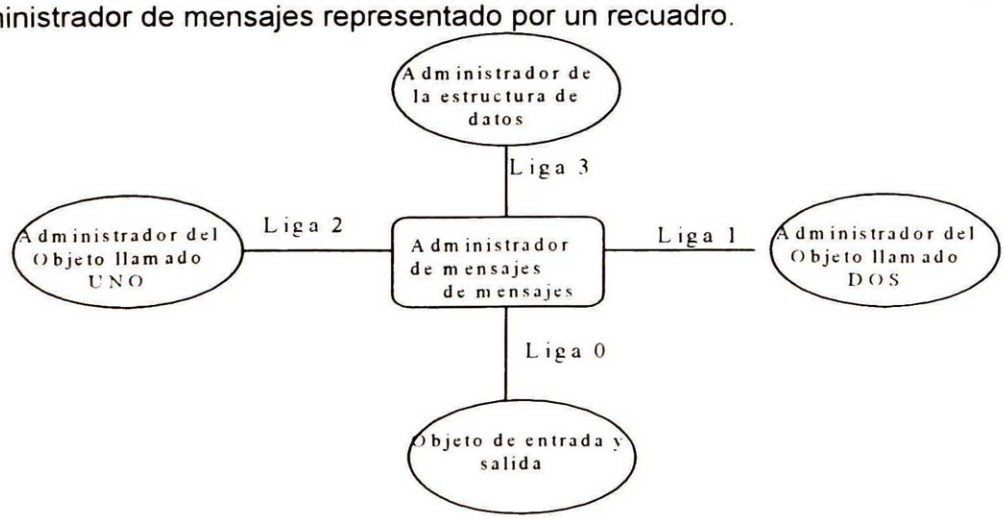


Figura 24. Esquema de comunicación entre objetos

A continuación se muestra el código completo del programa :

```
#INCLUDE "OBJETO.INC"
#INCLUDE "hostio.inc"

PROC COMUNICA(CHAN OF SP fs, ts, []INT memory)
  #USE "HOSTIO.LIB"
  #USE "CONVERT.LIB"
  #USE "OBJETO.LIB"

  IMPL OBJECT IMPRIME IS INT IDEN
    METHOD IMPRIME::ENTERO(INT X)
      SEQ
        PROC.ID,PROC.D.ID:=0,1
        DATO.ID,STRNUM:=0,0
        so.write.int(fs,ts,X,10)
      :
  :

  STRUCT STRNAME IS INT R1 ; INT R2:

  IMPL OBJECT UNO IS INT IDEN
    METHOD UNO::RECIBE()
      INT X :
      SEQ
        SEQ
          PROC.ID,PROC.D.ID:=2,1
          DATO.ID,STRNUM,X:=0,0,13
          STRNAME_R1:=X
          X:=STRNAME_R1
          PROC.ID,PROC.D.ID:=2,0
          OPER:=0
          IMPRIME_ENTERO(X)
    :
  :

  IMPL OBJECT DOS IS INT IDEN
    METHOD DOS::RECIBE()
      INT X :
      SEQ
        SEQ
          PROC.ID,PROC.D.ID:=3,1
          DATO.ID,STRNUM,X:=1,0,18
          STRNAME_R2:=X
          X:=STRNAME_R2
          PROC.ID,PROC.D.ID:=3,0
          OPER:=0
          IMPRIME_ENTERO(X)
    .
  :

  PROC EJECUTOR()
    SEQ
      WHILE TRUE
        SEQ
          OPER,PROC.D.ID:=0,2
          PROC.ID,DATO.ID,STRNUM:=4,0,0
          UNO_RECIBE()
          OPER,PROC.D.ID:=0,3
          PROC.ID,DATO.ID,STRNUM:=4,0,0
          DOS_RECIBE()
  :
  SEQ
    SKIP

:
```

VI.13. Ejemplo de declaración de objetos

En el siguiente ejemplo se utilizan tres tipos de objetos :

a) Punto : representado por dos enteros.
b) Rectángulo : Utiliza cuatro objetos punto.
c) Salida : objeto que ofrece el servicio de salida a vídeo.

El objeto Rectángulo solicitará servicios del objetos punto para definir sus cuatro coordenadas.

El constructor del Rectángulo simplemente creará cuatro instancias del objeto punto y asignará los valores de sus coordenadas.

El objeto rectángulo sólo tendrá dos métodos, el primero se encargará de enviar al objeto de salida, la petición de impresión de sus puntos, el segundo método, sumará un entero a sus respectivos puntos.

En el trabajo no se detallan técnicas de análisis y diseño orientado a objetos, aunque muchos autores han propuesto diferentes alternativas para el desarrollo de sistemas con orientación a objetos.

Las más conocidas son :

i) Análisis y Diseño orientado a objetos de Booch[25].
ii) Análisis y Diseño orientado a objetos de Coad/Yourdon[23][24].
iii) Object Modeling Technique (OMT)[27]

Existen otros autores como Bertrand Meyer[26] donde discute algunos conceptos en el análisis y diseño orientado a objetos en lenguajes concurrentes, sin embargo la metodología que utilicemos debe detallar, las interacciones entre objetos, su comportamiento concurrente y el conjunto de protocolos a los que responden y ofrecen sus servicios.

Las técnicas de Análisis y diseño orientado a objetos ayudan a modelar el mundo real y así por medio de un lenguaje orientado a objetos lo podemos representar fielmente, en este caso el lenguaje da la nueva alternativa de explotar el paralelismo implementado con procesadores TRANSPUTER.

VI.14. Esquema básico del problema

En la figura 24 se muestra la interacción entre los distintos objetos a través del administrador de mensajes, en este caso los objetos punto actúan como servidores de datos al objeto rectángulo.

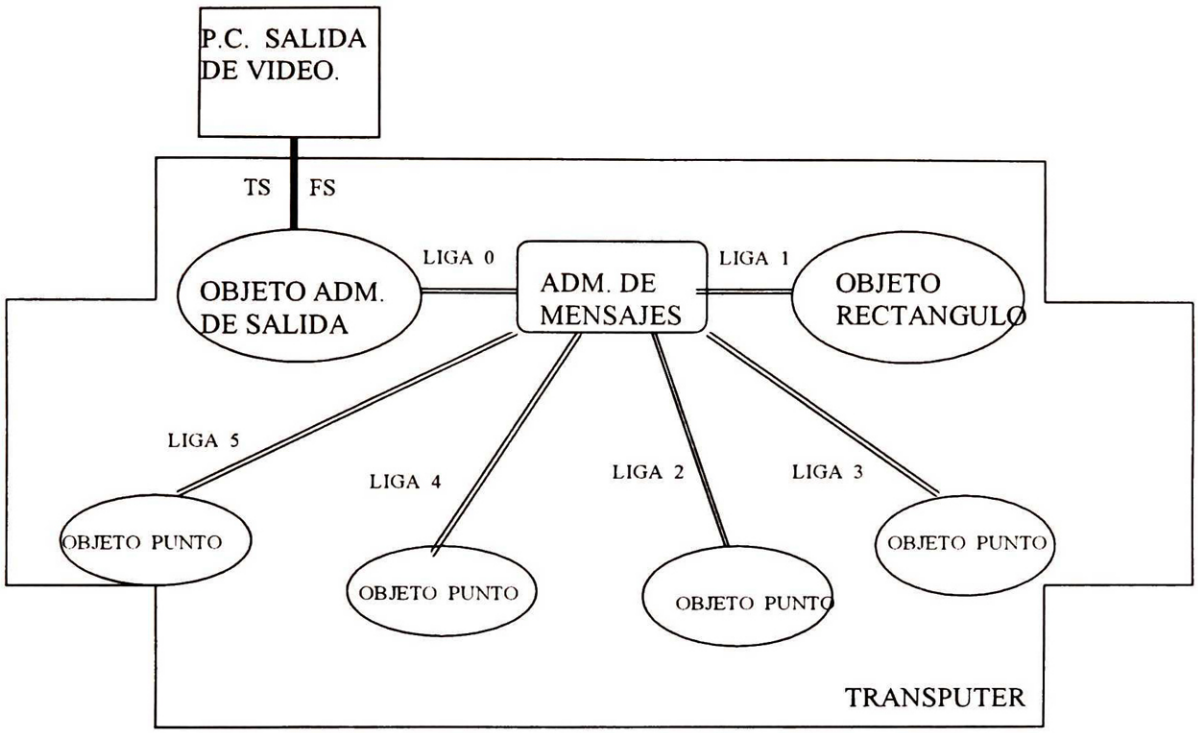


Figura 25 Objetos y el TRANSPUTER

El objeto punto es definido como:

```
IMPL OBJETO  PUNTO IS INT x; INT y

  CONSTRUCT:: PUNTO (INT x , INT y)
    SEQ
      Punto_x:= x
      Punto_y:= y
  :

  METHOD  PUNTO :: IMPRIME()
    INT Auxx,Auxy :
    SEQ
      Auxx=PUNTO_x
      Auxy=PUNTO_y
      IMPRIME_PUNTO(Auxx,Auxy)
  :

  METHOD PUNTO::suma(INT val)
    SEQ
      PUNTO_x := PUNTO_x + val
      PUNTO_y := PUNTO_y + val
  :
:
```

El objeto punto tiene:

a) Un constructor, que le asigna los valores que almacenará el objeto.
b) El objeto punto solo consta del método imprime que envía los dos valores enteros del punto al objeto IMPRIME.
c) Un método llamado suma, que adiciona un entero a su dos datos.
d) Recordemos que el objeto tiene dos método por default, que son la actualización y la consulta de datos.

El objeto Rectángulo

```
IMPL OBJETO RECTÁNGULO IS INT IDEN
  CONSTRUCT::RECTANGULO(INT x1,INT y1,INT x2,INT y2,INT x3,INT y3,INT x4,INT y4)
    SEQ
      PUNTO no(x1,y1);
      PUNTO so(x2,y2);
      PUNTO ne(x3,y3);
      PUNTO se(x4,y4);
:
  METHOD RECTÁNGULO :: IMPRIME()
    INT Auxx,Auxy :
    SEQ
      Auxx,Auxy :=no_x,no_y
      no_IMPRIME(Auxx,Auxy)
      Auxx,Auxy :=so_x,so_y
      so_IMPRIME(Auxx,Auxy)
      Auxx,Auxy :=ne_x,ne_y
      ne_IMPRIME(Auxx,Auxy)
      Auxx,Auxy :=se_x,se_y
      se_IMPRIME(Auxx,Auxy)
  :
  METHOD RECTÁNGULO :: SUMA(INT val)
    INT Auxx :
    SEQ
      no_SUMA(val)
      so_SUMA(val)
      ne_SUMA(val)
      se_SUMA(val)
  :
:
```

Dentro del constructor se instancian cuatro objetos punto, que contienen los valores de las cuatro coordenadas que representan a un rectángulo (ne,se,no,so).

Este planteamiento sólo es factible para este ejemplo, ya que si creamos otra instancia de un objeto rectángulo tendríamos que manipular diferentes nombre para las otras instancias del objeto punto.

Esta forma de definir objetos dentro del constructor no es factible, ya que existirían colisiones con otras instancias del objeto, en este ejemplo se empleó así para ejemplificar la comunicación entre objetos.

La idea general en el manejo de los métodos punto desde el objeto rectángulo es ejemplificativa de la solicitud de servicios. Por ej. la impresión de las esquinas del objeto rectángulo.

Finalmente tendremos el objeto cuyo servicio que ofrece es el de impresión:

```
OBJETO IMPRIME IS INT IDEN

  METHOD IMPRIME::PUNTO(INT Auxx,INT Auxy)
    SEQ
      so.write.int(fs,ts,Auxx,2)
      so.write.int(fs,ts,Auxy,2)
  :
:
```

IDEN : Es un identificador de los objetos.

La tarea del proceso de impresión es muy sencilla: envía hacia la PC los enteros correspondientes a los puntos.

VI.15. Conclusiones

Se mencionó acerca del entorno de desarrollo y se explicó la utilización de las nuevas construcciones del lenguaje OBOCCAM, como son, las que involucran a la definición del objetos y las invocaciones a los servicios que estos ofrecen.

También en forma general se ha descrito el enfoque que involucra la extensión y el uso de técnicas concurrentes orientadas a objetos , todo esto, con el propósito de demostrar cada uno de los mecanismos que fueron propuestos al inicio del trabajo, no cabe duda, que este es el inicio de un nuevo lenguaje que fácilmente se puede mejorar y adicionarle otras funcionalidades como es la herencia.

CONCLUSION GENERAL

La idea central de tener un lenguaje que sea ejecutado fácilmente por redes de procesadores paralelos y además, facilite la utilización de la programación orientada a objetos es bastante ambiciosa y el resultado que se ha obtenido ha sido OBOCCAM.

OBOCCAM probablemente se encuentra en sus primeros intentos de satisfacer esta idea, sin embargo, ya es una realidad, como el hecho de manipular un conjunto de proposiciones que siguen el paradigma de objetos y generar con ellas programas paralelos que sean ejecutados en una red de procesadores del tipo TRANSPUTER.

Por otra parte, en el trabajo se muestra la teoría en la que se desarrolló la construcción de la extensión, la forma en que se creó la similitud entre un objeto y su correspondiente proceso, la comunicación entre los distintos objetos, la sintaxis de las palabras clave, etc., con lo cual se comprueba lo complejo que resulta el contemplar todo el conjunto de variables implicadas en la extensión.

Así con lo anterior se tiene un lenguaje con el que se introducirían fácilmente en el mundo de la programación concurrente orientada a objetos y así, aplicar sus distintas técnicas en el desarrollo de sistemas, tales como :

-- Sistemas de inteligencia artificial
-- Sistemas de tiempo Real
-- Sistemas de información
-- Sistemas de cálculos complejos
-- Sistemas de despliegue gráficos
-- Entre otros.

Se han presentado cada uno de los conceptos y características que estrechamente se relacionan con OBOCCAM , los cuales, con el paso del tiempo, serán mejorados mediante la optimización de sus mecanismos.

Las limitaciones que se han encontrado en el trabajo son:

i) No se pueden crear objetos dinámicamente.

ii) La relación entre objetos es estática.

iii) No se permite la herencia.

iv) No se explota totalmente el polimorfismo de funciones.

v) No se permite el retorno de valores de los métodos como si estos fuesen funciones.

vi) Utiliza una sobrecarga de operadores limitada.

Aunque en general se cubren todos los puntos del trabajo, todavía en algunos de ellos hay que trabajarlos y tener resultados más favorables a la extensión.

En el desarrollo del preprocesador se utilizaron herramientas de compiladores como LEX y YACC, por lo que su núcleo básicamente es un autómata de pila, la manipulación de la gramática por medio de YACC nos facilita la realización de modificaciones sobre la extensión, por lo que su mantenimiento resulta bastante flexible.

Finalmente hemos presentado a OBOCCAM, el cual es un lenguaje que abarca áreas, como :

1.- Arquitectura paralelas

2.- Programación Concurrente ; y

3.- Programación orientada a objetos.

Este conjunto de conocimientos son el motivo que me llevo a trabajar en el desarrollo de OBOCCAM y una vez que he trasmitido las experiencias, dejo un precedente de mi trabajo.

GLOSARIO

ANCHO DE BANDA: Es la capacidad de transmisión de datos que tiene un medio físico de comunicación.

BUFFER: Localidades de memoria para almacenar datos temporalmente, se ocupa para los procesos, cuando estos utilizan primitivas de comunicación por paso de mensajes, para almacenar datos, que provienen de otro y así no se bloqueen por falta de lugar para almacenar los datos.

BUSY-WAITING: Primitiva de sincronización que cae en el conjunto de primitivas que utilizan variables compartidas.

CALENDARIZADOR: Es aquel mecanismo que puede ser implementado en software o hardware, el cual tiene como función organizar el uso del procesador para diferentes tareas(procesos) y darles un tiempo a cada una, este mecanismo puede tener diferentes tipos de políticas y prioridades.

CAMBIO DE CONTEXTO: Llámese así, al conjunto de pasos que se deben realizar cuando la ejecución de un proceso se finaliza, o su rebanada de tiempo se haya acabado y sea necesario su remplazo por otro proceso, los pasos más importantes son: grabar la información necesaria, para cuando le toque nuevamente el turno al proceso que se estaba ejecutando y así pueda seguir en lugar donde estaba, y además poner en ejecución el proceso que estaba en espera, todo esto se logra por medio de calendarización de procesos.

COBEGIN-COEND: Es una especificación de concurrencia para procesos, esta representación se utiliza en Pascal concurrente realizado por Brinch Hansen.

COMUNICACIÓN ENTRE PROCESOS: Es la transferencia de información entre procesos, esta se puede realizar de dos formas por paso de mensajes o por variables compartidas.

CONDICIONES DE SINCRONIZACIÓN: Es la condición que es requerida ya sea para accesar a una variable compartida o un recurso que también es compartido, esta únicamente se presenta cuando se utilizan técnicas de sincronización con variables compartidas.

EXCLUSIÓN MUTUA: Es la condición necesaria que requiere un proceso para tener exclusividad de un recurso que puede ser compartido y por lo cual tiene accesos concurrentes.

DEPURACIÓN: Es la acción de revisar programas y los efectos que estos produzcan en el sistema de cómputo, todo esto para ver su funcionamiento en forma detallada y sobre todo en búsqueda de posibles errores.

ENCAPSULAMIENTO:(Ocultamiento de la información) Identifica y separa los aspectos externos de un objeto de los detalles de su estructura interna. Los aspectos externos pueden ser accesibles por otros objetos, en cambio los internos son inaccesibles para ellos.

FORK-JOIN: Forma de poder especificar concurrencia en algún lenguaje de programación, su definición puede verse en el presente trabajo, en la sección de especificación de concurrencia.

GUARDIA: Variable Booleana que condiciona la ejecución de alguna sentencia. Por lo regular aparece en lenguajes que utilizan técnicas de sincronización por paso de mensajes, p.ej OCCAM.

HOST: Termino que recibe en una red de computadoras una computadora anfitrión.

INMOS: Empresa dedicada a la fabricación de componentes electrónicos, es la que se dedica a construir los procesadores TRANSPUTER.

INTERRUPCIONES: Llamadas por un proceso a otro o al administrador del sistema para interrumpir la ejecución de un proceso o la ejecución de una tarea por el procesador, existen diferentes tipos de interrupciones.

KERNEL (núcleo): Es la porción de código del sistema operativo, la cual se encarga en términos generales de la administración de procesos, como son: la interrupción, la creación, la destrucción, el cambio de contexto y otras acciones correspondientes a los procesos, que en un determinado momento se están ejecutando en un sistema de multiprogramación o multiproceso.

MAPEADO POR MEMORIA: Se dice que un recurso o un servicio es mapeado por memoria, cuando para accesarlo o solicitarlo respectivamente, es necesario referirse a él como si fuera una dirección de una localidad de memoria, además esta dirección es reservada dentro del rango de direccionamiento que tiene el procesador.

MICROOPERACIÓN: Operación básica realizada a bajo nivel en un sistema de cómputo, por lo regular un conjunto de ellas realizan una instrucción básica de un computador.

MIMD (Múltiple flujo de instrucciones con múltiple flujo de datos): Una Clasificación de sistemas paralelos basados en multiprocesamiento.

MULTICOMPUTADORAS: Varios sistemas de cómputo con las mismas o diferentes características, interconectados de tal manera para que por cooperación entre ellos tengan la facultad de procesar en paralelo.

MULTIPROGRAMACIÓN: Características de algunos sistemas de cómputo, los cuales permiten tener en ejecución a más de un programa.

OCCAM: Lenguaje concurrente que tiene la capacidad de que en él se pueda realizar programación paralela, además utiliza el paso de mensajes como primitivas para la comunicación y sincronización de procesos.

PASO DE MENSAJES: Llámese así al conjuntos de primitivas que lo utilizan para comunicación y sincronización de procesos.

PRIMITIVAS DE SINCRONIZACIÓN: Son mecanismos básicos para la comunicación y sincronización de procesos, existen dos tipos: por paso de mensajes y por variables compartidas.

PROCESADOR DE CANAL VIRTUAL: Existe una técnica utilizada en la familia de TRANSPUTER T9000, en la cual se multiplexan/demultiplexan canales entre procesos, esto es manejado por una parte de hardware llamado procesador de canal virtual (PCV), este es hábil para multiplexar un número de canales virtuales sobre un enlace físico. El termino virtual se refiere a que los procesos pueden utilizar varios canales como si fueran reales y en verdad únicamente es la conmutación de uno.

PROCESAMIENTO VECTORIAL: Forma de procesamiento de datos que emplean ciertos procesadores de ciertas arquitecturas de computadoras , en especial las supercomputadoras como la CRAY, en si se trata de ejecutar las operaciones por medio de vectores (arreglos de datos) los cuales contienen los datos, en un ciclo de reloj.

PROGRAMACIÓN CONCURRENTE: Es una metodología de diseño y programación, en la cual el sistema a ser construido es visto como una colección de procesos que se ejecutan concurrentemente.

PROGRAMACIÓN PARALELA: Es una metodología de diseño y programación, en la cual el sistema a ser construido es visto como una colección de procesos que se ejecutan paralelamente, en sí, es un caso de la programación concurrente únicamente que con la diferencia de que si se quiere que exista paralelismo debe de haber un conjunto de procesadores con la característica primordial de poder operar en paralelo y cooperar entre ellos.

PROGRAMAS SECUENCIALES: Es aquél en el cual, la ejecución de las sentencias que lo componen, para realizar su objetivo para el cual es construido, es en forma secuencial. Es decir que sin importar las diferentes formas de sentencias de bifurcación que contenga siempre se podrá representar en una forma secuencial para su ejecución.

REBANADAS DE TIEMPO: Es la fracción de tiempo que le corresponde a un proceso o a una tarea para poder ser ejecutados por el procesador.

RECURSO: Son todos aquellos dispositivos, tanto periféricos como internos, que tiene una computadora, los cuales pueden ser ocupados por algún proceso que se está ejecutando en el sistema.

SECCIÓN CRÍTICA: Es aquella variable o conjuntos de ellas o un recurso el cual puede ser accesado concurrentemente por más de un proceso.

SEMÁFOROS: Una primitiva de sincronización de procesos basada en variables compartidas.

SISTEMAS DE MULTIPROCESAMIENTO: Son aquellos sistemas de cómputo constituidos por más de un procesador y por lo cual tiene la capacidad de procesar en paralelo y poder ejecutar programas distribuidos.

SISTEMAS DE MULTIPROGRAMACIÓN: Son aquellos sistemas de cómputo con ejecutan varios programas al mismo tiempo.

SISTEMAS PARALELOS: Son aquellos que tienen la capacidad de procesar en paralelo.

TRASLAPE DE PROCESOS: Existe cuando dos o más procesos accesan al mismo tiempo en a un recurso compartido y por lo cual pueden ocasionar resultados inesperados.

SUPOSICIÓN DE PROGRESOS FINITOS: Suposición en la cual se refiere a que toda ejecución de un proceso concurrente siempre será en favor o positiva del objetivo que se desea realizar y nunca en un sentido negativo.

TIMER: Variable que contiene un valor incremental y cuyo incremento se relaciona con el reloj del sistema.

TRANSPUTER: Procesador del tipo RISC que puede interconectarse fácilmente y así generar sistemas paralelos, las características del procesador permiten la programación concurrente.

VARIABLE COMPARTIDA: Es aquella variable que es accedida por más de un proceso en la ejecución de un programa concurrente.

BIBLIOGRAFÍA

[1].-ANDREW S. TANEMBAUM; STRUCTURED ÇOMPUTER ORGANIZATION PRENTICE HALL. 1990.

[2].-DR. U VILLANO; TRANSPUTER AND PARALLEL ARQUITECTURES MENSSAGE-PASSING DISTRIBUTED SYSTEMS; ED. ELLIS HORWOOD LIMITED. 1991.

[3].-L.ALTAMIRANO ROBLES,S.PFLEGER; INFORME TÉCNICO: ARQUITECTURA RISC Y LA TRANSPUTER; CINVESTAV, DPTO. DE ING. ELÉCTRICA. 1991.

[4].-GREGORY R.ANDREWS; CONCURRENT PROGRAMMING PRINCIPLES AND PRACTICE; THE BENJAMIN/CUMMINGS PUBLISHING COMPANY INC. 1991.

[5].-INMOS,CSA COMPUTER SYSTEMS ARCHITECTS; TRANSPUTER EDUCTIONAL KIT; PUBLISH INMOS INC. 1990.

[6].-COMPUTER SYSTEMS ARCHITECTS; TRANSPUTER EDUCATIONAL KIT WORKBOOK ;PUBLISH INMOS INC. 1990.

[7].-AKINORI YONEZAWA,MARIO TOKORO; OBJECT-ORIENTED CONCURRENT PROGRAMMING; THE MIT PRESS; 1987.

[8].-JOSE OSCAR OLMEDO A.; EL LENGUAJE C++ CONCURRENTE SINOPSIS Y EJEMPLOS; CINVESTAV DEPTO. DE ING. ELÉCTRICA; 1993.

[9].-NARAIN GEHANI, ANDREW D. MCGETTRICK; CONCURRENT PROGRAMMING; ADDISON WESLEY PUBLISHING COMPANY; 1988.

[10].-C.A.R. HOARE; COMMUNICATING SEQUENTIAL PROCESS; COMMUNICATION OF ACM, AUGUST 1978,VOL.21,NUMBER 8 PP 666-677.

[11].-BRINCH HANSEN; DISTRIBUTED PROCESS: A CONCURRENT PROGRAMMING CONCEPT; COMMUNICATION OF ACM, NOVEMBER 1978, VOL. 21 NUMBER 11, PP 934-941.

[12].- COMPUTER SYSTEMS ARCHITECTS; A TUTORIAL INTRODUCTION TO OCCAM PROGRAMMING:OCCAM 2 TOOLSET, USER MANUAL; PUBLISH INMOS INC. 1989.

[13].- AKINORI YANEZAWA; ABCL AN OBJECT-ORIENTED CONCURRENT LANGUAGE THEORY, LANGUAGE,PROGRAMMING,IMPLEMENTACTION AND APPLICATION. MIT PRESS SERIES IN COMPUTER SCIENCE

[14].-DENIS CORONEL:TOWARD A METHOD OF O-O CONCURRENT PROGRAMMING COMMUNICATION OF ACM,SEPTEMBER 1993/VOL 36, NUM 9.

[15].-BERTRAND MEYER: SYSTEMATIC CONCURRENT O-O PROGRAMMING COMMUNICATION OF ACM,SEPTEMBER 1993/VOL 36, NUM 9.

[16].-BRUNO ACHAVER; THE DOWL DISTRIBUTED O-O LANGUAGE COMMUNICATION OF ACM,SEPTEMBER 1993/VOL 36, NUM 9.

[17].-MURAT KARAORMAN AND JOHN BRUNO: INTRODUCING CONCURRECY TO SEQUENTIAL LANGUAGE; COMMUNICATION OF ACM, SEPTEMBER 1993 VOL 36, NUM 9.

[18].-JOSE OSCAR OLMEDO A.: PLM-86 CONCURRENTE; TESIS DE MAESTRIA SECCION COMPUTACIÓN DEPTO. DE ING. ELECTRICA. CINVESTAV IPN.

[19].-B. HENDERSON-SELLERS; A BOOK OBJECT-ORIENTED KNOWLEDGE , OBJECT ORIENTED ANALISIS,DESIGN AND IMPLEMENTATION: A NEW APPROACH TO SOFTWARE ENGINEERING; PRENTICE HALL.

[20].-TANNENBAUM A.S. ; COMPUTER NETWORKS; PRENTICE HALL, ENGLEWOOD CLIFFS N.J., 1981.

[21].-DOUGLAS COMER; INTERNETWORKING WITH TCP/IP PRINCIPLES, PROTOCOLS AND ARCHITECTURE; PRENTICE HALL VOL 1. 1988.

[22].- DOUGLAS COMER, DAND L. STEVENS; INTERNETWORKING WITH TCP/IP CLIENT-SERVER, PROGRAMING AND APPLICATIONS; PRENTICE HALL VOL 1 1988.

[23].- PETER COAD, EDUARD YOURDON ; OBJECT ORIENTED ANALISYS: YOURDON PRESS, PRENTICE HALL , 1991.

[24].- PETER COAD, EDUARD YOURDON ; OBJECT ORIENTED DESIGN: YOURDON PRESS, PRENTICE HALL , 1991.

[25].- BOOCH G. ; ORIENTED OBJECTS DESIGN ; BENJAMIN CUMMINGS ; 1990.

[26].- BERTRAND MEYER ;SYSTEMATIC CONCURRENT OBJECT ORIENTED ;COMMUNICATIONS OF THE ACM, VOL 36 NUM. 9. SEPTEMBER 1993.

[27].- TOM GOODELL; REAL TOOLS FOR THE REAL WORLD; POPKIN SOFTWARE & SYSTEMS INC.; VOL 4 NUM. 2 1995.

Los abajo firmantes, integrantes de jurado para el examen de grado que sustentará el **Lic. Raúl Hernández Cruz**, declaramos que hemos revisado la tesis titulada:

"Oboccam: Occam basado en objetos", consideramos que cumple con los requisitos para obtener el grado de Maestro en Ciencias, con especialidad en Ingeniería Eléctrica.

Atentamente

Dr. Héctor Ruíz Barradas _____

Dra. Ana María Antonia Martínez Enríquez _____

Dr. Guillermo Benito Morales Luna _____