Contenido

Editorial

El Centro de Innovación y Desarrollo Tecnológico en Cómputo (CIDETEC) surge en su momento con los objetivos de consolidar la operación de la infraestructura de cómputo del Instituto, de desarrollar nuevas aplicaciones de la misma, de aprovechar las ventajas tecnológicas para incidir en el proceso productivo de bienes y servicios y de impartir cursos de capacitación, actualización, especialización y superación académica y profesional en el campo de la ingeniería de cómputo.

Con el fin de promover la superación profesional de los recursos humanos en el área de la Computación, se determinó incluir en el plan de trabajo 1998-2000 del Centro la oferta de un programa de Maestría, para ser impartida a partir del semestre de primavera de 1999. Esta determinación se fundó tanto en la necesidad de complementar y fortalecer las actividades que el Centro realiza en materia educativa, de investigación científica, de innovación tecnológica y de operación y mantenimiento de los sistemas y equipos de cómputo con que cuentan las distintas áreas, escuelas y centros del Instituto, como en la de integrar a la oferta educativa del Instituto Politécnico Nacional aquellas áreas que no se desarrollan en la medida de lo deseable en las escuelas y centros que ofrecen programas de posgrado en computación. Este objetivo no se pudo realizar en su momento debido a diversos cambios en la dirección del Centro, y por la falta de recursos humanos calificados en medida suficiente para implantar un programa de posgrado.

En este momento, sin embargo, el CIDETEC ya se encuentra en posibilidades de concretar estos objetivos. La propuesta que actualmente esta desarrollando el Centro retoma el objetivo antes citado, revisado y actualizado con base en las necesidades tecnológicas de los sectores productivo y de servicios, como del mismo Instituto. En lineas generales, el objetivo final que se pretende lograr con esta propuesta es:

Formar recursos humanos en el área de la Ingeniería de Cómputo, así como especializar y actualizar a profesionales de la computación en esta área, que al egresar del programa contarán con los conocimientos y habilidades para desempeñar las siguientes actividades:

- Generar aplicaciones técnicas y científicas de la computación.
- Programar máquinas de alto rendimiento y aplicaciones de supercómputo.
- Implementar sistemas de medición y control computarizados para procesos industriales.
- Diseñar sistemas dedicados, tanto en lo que a hardware, como a software se refiere.
- Diseñar y programar sistemas de cómputo en el campo de la realidad virtual.
- Implementar sistemas de cómputo distribuido.

En el contexto de los países miembros del Tratado de Libre Comercio, así como en la Comunidad Europea, se distinguen las curricula de Informática, Ciencias de la Computación, Ingeniería de Software e Ingeniería de Cómputo como áreas determinadas para objeto de estudio. En particular, la Association for Computing Machinery (ACM) y la Computer Society del Institute for Electrical and Electronic Engineers (IEEE-CS) definen a esta última como el análisis, diseño, construcción y la aplicación de computadoras y de sistemas digitales para la generación tanto de nuevas máquinas, como de aplicaciones dedicadas de la computación. La disciplina involucra tanto el hardware como el software, así como la interacción entre ambos. Un estudio comparativo de los programas de posgrado ofrecidos en nuestro país que se anexa a esta propuesta muestra claramente que la oferta en esta área es sumamente pobre. El diseño de sistemas de cómputo dedicados y/o de propósito específico prácticamente no es cubierto en forma sistemática e integral. Así, aunque existen programas académicos relacionados con el diseño de sistemas digitales por una parte, y otros que se ocupan de la programación de aplicaciones técnicas y científicas, no existe un programa que integre estas áreas para aplicaciones desde medición, control de procesos, automatización industrial etc., hasta la generación de sistemas integrales dedicados en, por ejemplo, el diagnóstico médico.

Por otra parte, el creciente impacto de la educación no presencial en todas sus formas genera nuevas necesidades en materia de infraestructura computacional y de comunicaciones, así como en materia de programación en áreas tan computacionalmente intensivas como la realidad virtual. Estas nuevas aplicaciones requieren soluciones propias, no sólo por la importancia económica del mercado, sino por necesidades de control y óptimo aprovechamiento de la infraestructura sobre una base de disponibilidad cercana al 100%.

La propuesta que el CIDETEC esta desarrollando pretende, así, llenar un vacío de la oferta educativa no sólo del Instituto, sino del país, en este campo de creciente importancia.

La Técnica del Mapeo

M.en C. G. Abraham Mas Levario
M. en C. M. Aurora Segura Corona
Ing. Héctor Cortés León
Profesores del Centro de Investigación en Computación, IPN.

El presente artículo plantea la técnica del mapeo como una de las soluciones para direccionar memoria ubicada en una tarjeta de expansión que se encuentra conectada en una de las ranuras de expansión de las computadoras personales, la cual es la anfitriona y responsable del mapeo.

Conceptualización del Mapeo

Aquí se muestra la idea general de lo que es el mapeo de memoria, así como ciertas definiciones que permiten desarrollar los métodos matemáticos que cualifican la idea del mapeo.[4]

1.1 Descripción general del mapeo

La apariencia y el postulado del mapeo se muestran en la **figura 1-1**. La idea básica es hacer que cualquier acceso al segmento, esté reflejado por medio del mapeador en la memoria mapeada. La memoria mapeada podría estar constituida por ocho dispositivos 6164; la capacidad de estos dispositivos es de 8 Kbytes, que dan un total de 64 Kbytes. Para ver un poco más de cerca el mapeo, supóngase que desde la PC se está direccionando el offset 0000 del segmento D, o sea en la dirección D000: 0000 del modo de operación real de la PC, y también supóngase que la PC no tiene memoria disponible físicamente en el segmento D, entonces, cuando desde un programa de la PC se escriba o se lea un dato en esa dirección (12), la memoria mapeada es direccionada por el mapeador para recibir o dar el dato; y cuando se da esta circunstancia es cuando existe el mapeo.

1.2 Axiomas

Se presenta una relación de ciertas observaciones que evidencian con más detalle el fenómeno del mapeo como cualquier hecho que puede ser percibido. Las observaciones son esencialmente los siguientes axiomas:

1. Cualquier idea o conjuntos de ideas pueden ser representadas con matemáticas con la finalidad de poder cualificar y cuantificar el objeto mental al que se refiere o refieren dichas ideas.
2. Cualquier objeto o conjunto de objetos computacionales pueden ser organizados con la cualificación y cuantificación del objeto mental al que se refieren tales ideas.

Corolario 1. En vista de lo anterior a cualquier objeto computacional se le puede atribuir cierto concepto cuando uno puede percibirlo.

3. A cualquier objeto computacional se le puede asignar un nombre según un criterio que tenga relación con la zona donde se aplique, por lo tanto a éste lo llamamos mapeador.
4. La primera acción creada en un mapeo es la comunicación entre el entorno de la PC y el mapeador.

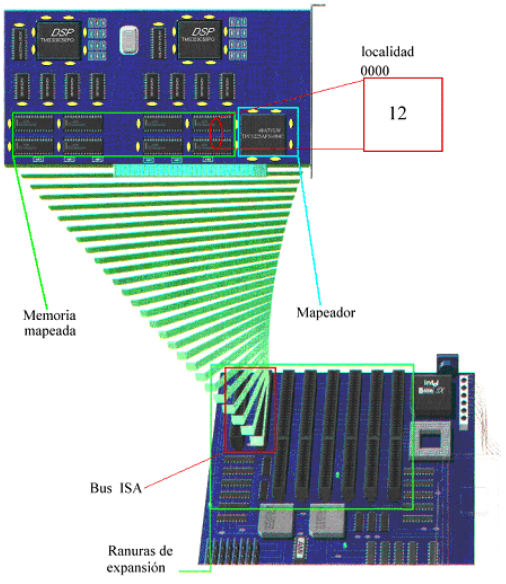

Figura 1-1. Tarjeta del sistema de memoria mapeada implementado con FPGA para un arreglo lineal de dos DSP TMS320C50.

5. La segunda acción creada en un mapeo es la comunicación entre el segmento D del entorno de la PC y la memoria mapeada.
6. El primer factor creado en una comunicación es el fenómeno de localización de espacios, llamado en esta tesis direccionamiento.

1.3 Teoremas que fundamentan el diseño teórico del mapeador

Teorema 1. Las líneas más significativas del direccionamiento del bus ISA son los factores que contribuyen a que el mapeador establezca la correspondencia unívoca entre el segmento D de memoria de la PC y la memoria mapeada.

Teorema 2. Las líneas menos significativas del direccionamiento del bus ISA son parte de la correspondencia unívoca entre el segmento D de memoria de la PC y la memoria mapeada.

1.4 Primer acercamiento matemático al mapeo

El fenómeno del mapeo es esencialmente una circunstancia de localización por reflejación desde el punto de vista de la tarjeta coprocesadora, y además es una circunstancia de localización por correspondencia desde el punto de vista del mapeador quien genera la comunicación entre dos entidades completamente diferentes.

Bien, cuando tenemos el mapeo como una circunstancia de localización por correspondencia se está usando el método matemático llamado "correspondencia" y en base a su definición se puede iniciar el estudio del mapeo.

La definición de correspondencia en el campo de estudio de la teoría de conjuntos, es: la asociación correspondiente de los elementos de un conjunto con los elementos de otro.

Entonces, de acuerdo con lo anterior supóngase que en el mapeo se distinguen dos conjuntos, uno es el conjunto llamado "segmento D" de la memoria en el modo real de la PC y el segundo es el conjunto llamado "memoria mapeada". Ambos conjuntos contienen 64 Kbytes de localidades. En donde, las localidades del segmento D son expresadas mediante los símbolos que van desde D0000:0000 hasta D0000:FFFF, y las localidades de la memoria mapeada desde 0000 hasta FFFF. La representación esquemática y correspondencia postulada de los dos conjuntos está en la **figura 1-2**.

La correspondencia que se da en el mapeo existe sólo para un par de elementos asociados en un determinado tiempo t_n, donde n es cualquier número mayor que cero. Además este tipo de correspondencia se llama "unívoca", porque todo elemento del segmento D se relaciona con uno, y sólo uno del conjunto memoria mapeada, y de este modo la asociación es continua desde D000: 0000→0000 hasta D000:FFFF→FFFF.

Con los resultados anteriores y con el concepto de "relación", se representa el mapeo en una forma matemática más compacta y manipulable para su estudio. Una relación se define como un conjunto de pares ordenados obtenidos por una correspondencia previamente establecida. Al observar la **figura 1-2** vemos que la correspondencia entre la localidad D000:0000 y la localidad 0000, representada por una flecha dirigida hacia la derecha, asocia un par de localidades indicadas como (D000: 0000, 0000), que es un par ordenado. Esto se cumple con cada una de las asociaciones anteriores, de modo que la relación de los conjuntos "segmento D" y "memoria mapeada" se muestra en el **listado 1**.

El conjunto indicado es la simbolización del mapeo para todos los

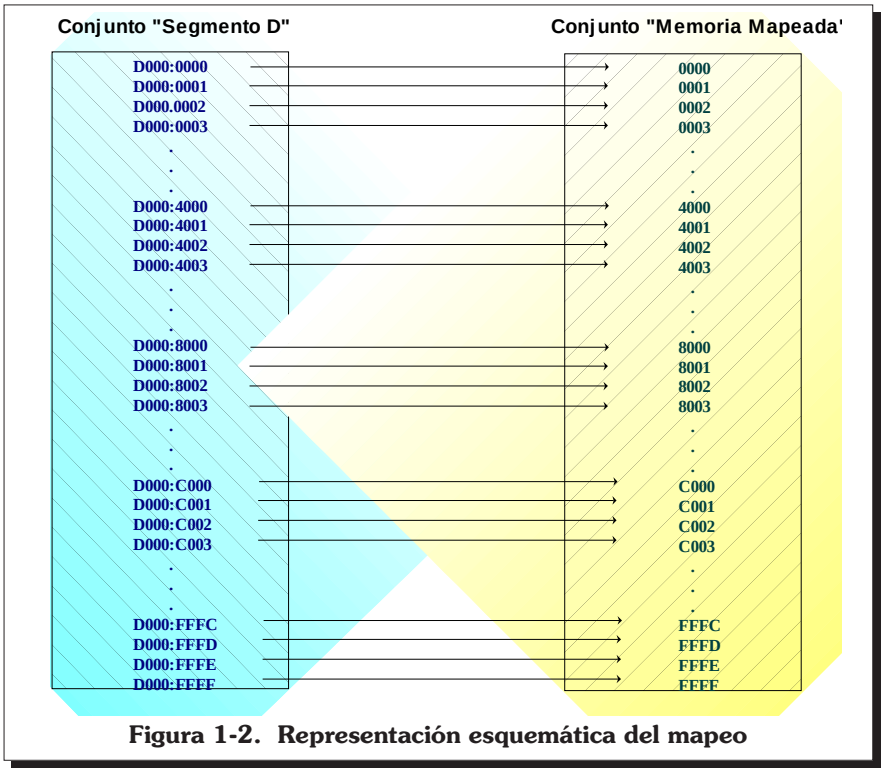

Figura 1-2. Representación esquemática del mapeo

```
{(D000:0000, 0000), (D000:0001, 0001), (D000:0002, 0002), (D000:0003, 0003),...
 (D000:4000, 4000), (D000:4001, 4001), (D000:4002, 4002), (D000:4003, 4003), ...
 (D000:8000, 8000), (D000:8001, 8001), (D000:8002, 8002), (D000:8003, 8003), ...
 (D000:C000,C000), (D000:C001,C001), (D000:C002,C002), (D000:C003,C003), ...
 (D000:FFFC,FFFC), (D000:FFFD,FFFD), (D000:FFFE,FFFE), (D000:FFFF,FFFF)
}
```

Listado 1. Relación de los conjuntos "segmento D" y "memoria

instantes en donde él exista. Hasta este punto se ve que cada offset del segmento D corresponde exactamente a una localidad cuyo número es el mismo que el offset del segmento D.

Finalmente, tomando en consideración que el mapeo es un conjunto de pares ordenados de números, tales que ninguna pareja de ellos tiene el mismo primer número, y que los segundos números de las parejas pueden ser obtenidos mediante una igualdad, que se puede simbolizar a su vez como la ecuación de la función que se establecerá más adelante.

Antes de establecer la función es necesario representar las direcciones del segmento D como DNNNN en lugar de D000:NNNN, donde cada "N" es cualquiera de los siguientes números en el sistema hexadecimal: "0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F". El motivo de lo anterior es que en el diseño digital es más común y práctico representar una dirección del bus de direcciones del modo real de las PC con cinco números hexadecimales para una posible manipulación algebráica.

De acuerdo a la observación de que la magnitud del offset del segmento D es igual a su correspondiente en la memoria mapeada, se puede generalizar a todos los pares ordenados del conjunto del mapeo como (DNNNN, NNNN) para deducir que el segundo número del par es el offset que se direcciona en el segmento D, por lo tanto la ecuación de la función es:

$$f(DNNNN) = DNNNN - D0000 \qquad (1\text{-}1)$$

En donde NNNN es cualquier valor entre 0000h y FFFFh y el resultado de f(DNNNN) es el segundo número hexadecimal de cierto par ordenado de un mapeo. Cada una de las N de la función es un número hexadecimal y como tal puede ser representado con una notación binaria de cuatro dígitos mediante los respectivos valores de cuatro líneas de direcciones del bus, además supóngase que NNNN se puede representar como $N_{12}N_8N_4N_0$, en donde los subíndices de las N indican el subíndice de la línea de direcciones menos significativa al formar el número hexadecimal. Así que N_0 que es la que está más a la derecha se refiere al valor hexadecimal que forman el grupo de cada una de los magnitudes de las primeras cuatro líneas físicas del bus de direcciones, o sea, $N_0=(A_3A_2A_1A_0)$. Siguiendo lo anterior, el valor de la N_4 se obtiene con la siguiente igualdad: $N_4=(A_7A_6A_5A_4)$, por lo tanto N_8 y N_{12} se obtienen con las siguientes igualdades:

$N_8=(A_{11}A_{10}A_9A_8)$,
$N_{12}=(A_{15}A_{14}A_{13}A_{12})$,

respectivamente.

La resta de la **ecuación 1-1** es la operación algebráica que establece el mapeo en sí. Esto quiere decir que el valor en el cual la PC está direccionando en el "segmento D" no se utiliza en la correspondencia unívoca, más bien, es el factor que crea la correspondencia unívoca y que es la condición para que exista el mapeo. Se concluye por la anterior observación que para hacer un direccionamiento en la memoria mapeada sólo se usan las direcciones desde la A_0 a la A_{15} del bus de direcciones del bus ISA. Ahora, se sabe que para direccionar memoria en el modo real desde el bus ISA se usan veinte líneas de direcciones que van desde la A_0 a la A_{19} y hasta este punto sólo se ha justificado de un modo claro el empleo de dieciséis líneas (A_0, A_1, ..., A_{14}, A_{15}) en el mapeo. Entonces se postula que la resta de la **ecuación 1-1** es una aproximación y un modo implícito de decir que con las líneas A_{16}, A_{17}, A_{18}, y A_{19} se crea de algún modo el factor que contribuye al mapeo.

En las siguientes secciones se tratarán métodos complementarios a los anteriores para finalizar el estudio del mapeo.

1.5 El mapeador como parte de una red de interconexión

Se puede considerar a todo el conjunto de mapeadores de las posibles tarjetas de expansión que interactuen con el bus ISA como una red de interconexión dinámica. Una red de interconexión dinámica, desde el punto de vista de esta tesis, es una estructura física de direccionamiento por etapas; cada una de las etapas posee componentes básicos que se dedican a direccionar las memorias mapeadas de la tarjeta de expansión en el ambiente de las ranuras del bus ISA. Estos componentes son enlaces y conmutadores programables.

Si se considera que la PC es la única fuente de la red y que las memorias mapeadas son los destinos, entonces se puede ver que la red

está estructurada de tal manera que permite hacer una sola conexión de una fuente a un destino a la vez. Y si se supone que hay memoria mapeada en cada uno de los dieciséis segmentos de memoria de la PC, entonces la red tiene dieciséis posibles destinos. Como únicamente se cuenta con las veinte líneas (A_0, A_1, ..., A_{18}, A_{19}) del bus de direcciones del bus ISA para establecer una conexión de mapeo, por lo tanto se puede deducir que las líneas A_{19}, A_{18}, A_{17} y A_{16} forman parte del mecanismo de direccionamiento de las etapas de la red debido a que tales líneas indican uno de los dieciséis segmentos, o sea, uno de los dieciséis destinos.

Para establecer las características de la red se postulan una serie de suposiciones que se enumeran abajo.

1. Se usa una sola línea de las cuatro anteriores (A_{19}, A_{18}, A_{17}, A_{16}) en cada etapa para direccionar gradualmente un destino.
2. Como solo entra una línea en cada etapa, entonces los conmutadores de cada etapa deberán tener una sola entrada de programación, y además cada una de estas entradas de programación deberán estar conectadas conjuntamente con la línea que llega a esa etapa.
3. El conmutador solo puede conmutar hacia uno de dos estados posibles (estado 0 o estado 1) debido a la naturaleza binaria de la línea que se conecta a su entrada de programación. Por lo tanto, el conmutador programable hace una unión de la entrada al estado 0 si la línea trae un valor de 0 o una unión de la entrada al estado 1 si la línea tiene un valor de 1.
4. Como la red direcciona gradualmente un destino y además tiene únicamente una entrada, se concluye que los conmutadores programables de cada etapa tienen únicamente una entrada.

Ahora, si se considera que se tiene una relación de números binarios, a partir de los valores que se puedan leer con las cuatro líneas (A_{19}, A_{18}, A_{17}, A_{16}) de un direccionamiento, en cada uno de los segmentos. El bit más significativo de ese cuarteto es A_{19}, en la relación mostrada en la **tabla 1-1**, se percibe que, en esta línea, los primeros ocho valores son cero y en los últimos ocho son uno. El número de A_{18} cuando A_{19} es cero, cambia de cero en los primeros cuatro valores a uno en los siguientes cuatro valores, y cuando A_{19} es uno, A_{18} cambia de cero nuevamente en los siguientes cuatro valores hasta que finalmente es uno en los últimos cuatro valores; de este modo también se puede distinguir cómo va variando A_{17} con respecto a A_{18}, y cómo va variando A_{16} en relación con A_{17}.

TABLA 1-1
Tabla de números binarios

A_{19}	A_{18}	A_{17}	A_{16}
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

Se puede considerar que con la anterior regla se está indicando, que el número de salidas de cada etapa es según el número de grupos de ceros o unos que originen las anteriores líneas de direcciones. Así que como A_{19} origina dos grupos (un grupo de ocho ceros y un grupo de ocho unos), entonces la etapa que controla esta línea solo tiene dos salidas, y de ahí que esta etapa sólo tenga un conmutador programable. Se puede entonces suponer que la PC se conecta directamente a dicha etapa. La línea A_{18} origina cuatro grupos (dos grupos de cuatro ceros y dos grupos de cuatro unos), de modo que la etapa controlada por A_{18} tiene cuatro salidas, y para tener cuatro salidas se requiere dos conmutadores, por lo tanto la etapa controlada por A_{18} se conecta a las dos salidas de la etapa controlada por A_{19}. Finalmente, la etapa controlada por A_{17} se conecta a las cuatro salidas de la etapa controlada por A_{18}, y la etapa controlada por A_{16} se conecta a las ocho salidas de la etapa controlada por A_{17}. En la **figura 1-3** se muestra la estructura de la red de interconexión dinámica para el mapeo en cada uno de los segmentos.

Generalizando, la red consiste de n = N / A etapas, donde N es el número de salidas y A es el número de líneas usadas para establecer una conexión entre la PC y la memoria mapeada. Como N = 16 y A = 4, entonces n = 4 etapas. Cada etapa en la red consiste de un cierto número de enlaces y de conmutadores, en donde el número de enlaces y el número de conmutadores es el mismo de acuerdo a la posición de la etapa en la red. La etapa 1 que está controlada por el bit más significativo (A_{19}) de la dirección del destino tiene la posición cero, la etapa 2 que esta controlada por A_{18} tiene la posición uno, la que controla A_{17} es la etapa 3 y tiene la posición dos, y la que controla A_{16} es la etapa 4 y tiene la posición tres.

Cada etapa consiste de $m = 2^p$ enlaces y conmutadores, en donde p es el número de la posición de la etapa, en la red. Según lo visto anteriormente, la etapa 1 consta de un enlace y de un conmutador ($m = 2^0 = 1$), la etapa 2 consta de dos enlaces y

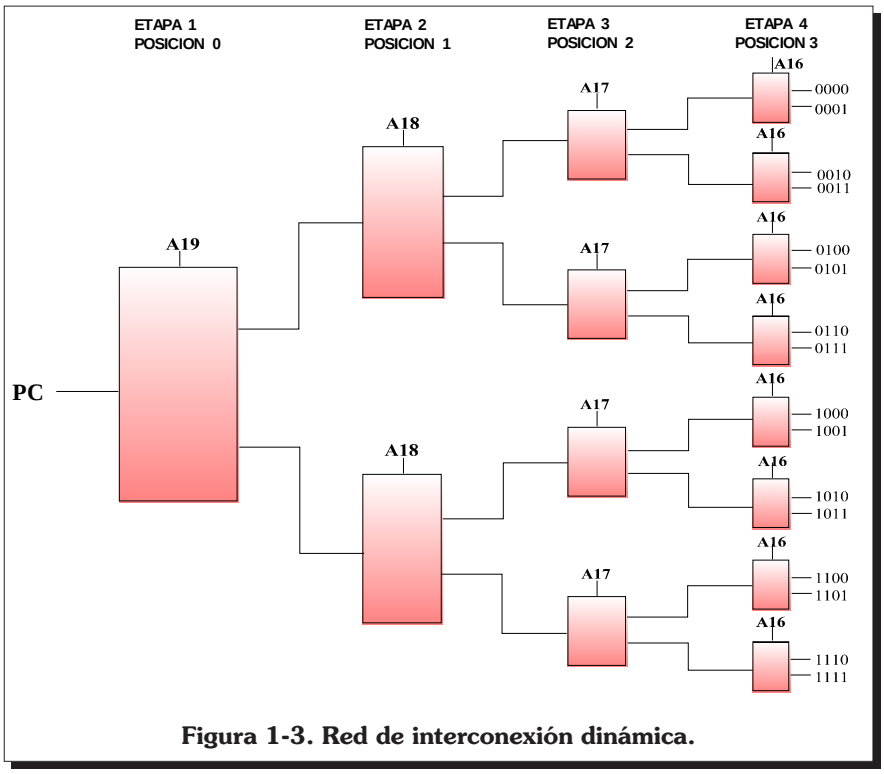

Figura 1-3. Red de interconexión dinámica.

dos conmutadores (m = 2^1 = 2), la etapa 3 consta de cuatro enlaces y cuatro conmutadores (m = 2^2 = 4) y la etapa 4 consta de ocho enlaces y ocho conmutadores (m = 2^3 = 8).

Con lo que respecta a los enlaces de cada una de las etapas de la red, se aprecia que estos son buses de direcciones que están formados por las primeras dieciséis líneas de direcciones (A_0, A_1, A_2, A_3, …, A_7, A_8, …, A_{12}, A_{13}, A_{14}, A_{15}) del bus ISA debido a que las ultimas cuatro (A_{16}, A_{17}, A_{18}, A_{19}) se emplean específicamente para controlar, a través de la red, el direccionamiento de una localidad de memoria mapeada.

1.6 Construcción del sistema de programación del mapeador

En base a los estudios cuantitativos y cualitativos anteriores se concluye que el mapeador requiere un sistema para programar a los conmutadores. Un sistema de programación en los sistemas digitales básicamente está integrado por expresiones Booleanas, las cuales determinan cómo debe operar el sistema según el estimulo recibido.

A partir de la figura 1-3 y del concepto de permutación para una red de interconexión se obtendrá para el sistema una expresión Booleana. Una permutación se refiere a la conexión de un conjunto de fuentes a un conjunto de destinos tales que cada fuente es conectada a un destino simple; al observar este concepto se ve que es congruente con lo aplicado anteriormente en la correspondencia unívoca. La representación simbólica de una conexión en este caso es la representación de un par ordenado. En vista de lo anterior y recordando que la red solo activa a la vez un mapeador, la permutación en este caso es [(S,D)] lo que significa que únicamente existe un mapeador en la red haciendo una conexión de una fuente "S" con un destino "D".

Como la permutación es esencialmente un par ordenado se puede encontrar una regla de igualdad que establezca como determinar el segundo miembro a partir del primer miembro. Esta regla (para los fines de este estudio) puede ser representada como una función Booleana siguiendo la suposición de que si se esta hablando de una conexión, implícitamente se esta tratando con un espacio direccionado y según con lo ya visto se puede establecer que S denota la representación binaria de una fuente, de modo que se puede establecer que S = $A_{19}A_{18}A_{17}A_{16}$ indica con cual destino se hace la conexión. Ahora, para tener consistencia el destino D también debe denotar una representación binaria con cuatro números binarios. Así que la permutación que indica que existe mapeo en el segmento D (A_{19}=1, A_{18}=1, A_{17}=0, A_{16}=1) es:

[(1101, 1101)] (1-2)

Si la función Booleana es F(S) y si además suponemos, en base a la figura 1-3, que cada etapa de la red de una conexión es una función Booleana parcial de F(S), entonces F(S) = $f_{19}f_{18}f_{17}f_{16}$. Estas funciones Booleanas parciales son básicamente funciones de conmutación; entonces, en general, cualquier permutación de una red puede ser representada en términos de un conjunto de funciones de conmutación. En vista de lo anterior, la determinación de F(S) en base a la magnitud de la fuente S, se encuentra a partir de una tabla de verdad simple, como la siguiente:

S				Conecta a	F(S)			
A_{19}	A_{18}	A_{17}	A_{16}	®	f_{19}	f_{18}	f_{17}	f_{16}
1	1	0	1	→	1	1	0	1

Tabla 1-2. Tabla de implementación de la función.

Cada una de las funciones de conmutación f_{19}, f_{18}, f_{17}, y f_{16}, por lo

tanto pueden ser expresadas como:

$f_{19} = A_{19}$,
$f_{18} = A_{18}$,
$f_{17} = \overline{A}_{17}$,
$f_{16} = A_{16}$.

La función Booleana, entonces, es $\mathbf{F(S) = A_{19}A_{18}\overline{A}_{17}A_{16}}$, lo que indica que para encontrar el destino se hace D = F(S). Esta función Booleana se cumple cada vez que se desea mapear memoria en el segmento D, o sea, cuando la permutación existente en la red de interconexión de mapeo es [(1101, 1101)], así que el mapeador para el segmento D debe integrar en su sistema de programación la implementación de la función a la cual se le puede nombrar como una función de mapeo de la red de interconexión. En general, se puede decir que la es la función sintética de las dieciséis posibles funciones de la red de interconexión de mapeo. La función F(S) además de indicar que destino se conecta con el fuente, indica a través de los valores de las líneas A_{19}, A_{18}, A_{17} y A_{16} cómo programar los conmutadores de cada etapa para direccionar gradualmente un destino.

1.7 Consideraciones prácticas del mapeador

Se puede suponer que el mapeador consiste de buses, dispositivos programables y dispositivos de programación. Los buses, además de estar constituidos por las primeras dieciséis líneas de direcciones del bus ISA, también contienen los datos del mismo bus el cual lleva en sí la información que direcciona las direcciones en la memoria mapeada.

Los dispositivos programables son los buffers que controlan la posibilidad de conexión de los buses a la memoria mapeada. Estos pueden ser implementados a través de circuitos TTL, como el dispositivo 74244 para la parte de direcciones y 74573 para la parte de datos de los buses, u otra posibilidad es la implementación de buffers en una FPGA.

Se puede decir que los dispositivos de programación son los circuitos que reciben tanto las líneas de control del bus ISA como las líneas de direcciones que generarán posteriormente la función de mapeo, y dependiendo de cómo se implemente la función de mapeo será la selección del dispositivo programable. La implementación puede llevarse a cabo a través de una ecuación Booleana en un dispositivo PLD tal como una GAL, o a través de una función programada en un dispositivo FPGA.

Si se pone de nuevo atención sobre la anatomía de la memoria mapeada (que se dio al inicio de este capítulo), podemos ver que la idealización de contar con un dispositivo físico de memoria con una capacidad de 64 Kbytes lineales realmente se obtuvo de la abstracción del conjunto de ocho dispositivos SRAM 6164 de 20 ns, lo que facilitó el estudio del mapeo. Ya que este dispositivo es de 8k X 8 bits posee 13 líneas de direcciones para direccionar una localidad entre 8192 disponibles, y además se postula usar la línea A_0 del bus de direcciones del bus ISA como un control para direccionar palabras de datos de 16 bits en lugar de palabras de 8 bits, entonces se aplicarán respectivamente las líneas A_1, A_2, hasta A_{13} de direcciones del bus ISA a las líneas A_0 hasta A_{12} del bus del mapeador; a su vez, estas últimas líneas se aplican a las entradas A_0 hasta A_{12} respectivamente de los dispositivos 6164.

En vista de lo anterior se tienen realmente cuatro bloques de memoria mapeada en lugar de uno, así que las líneas A_{14} y A_{15} se añaden a la función de mapeo, por lo tanto la función es esencialmente el mecanismo para decodificar la memoria mapeada, ya que dependiendo de los valores de las líneas A_{14} y A_{15} se mapea uno de los cuatro bloques de memoria.

Posteriormente se presentará la implementación del mapeador con dispositivos TTL y dispositivos GAL, y también se da la alternativa para implementarlo con la tecnología de FPGA.

Hasta este momento la función de mapeo no está totalmente construida, tiene los elementos básicos para contribuir al mapeo, pero por sí mismos no garantizan que la propuesta se logré adecuadamente.

Conclusiones

La formulación de los postulados condicionarón la definición del mapeo, así como el conocimiento de los axiomas de la ingeniería de cómputo, razón por la cual se presentó una descripción cualitativa y cuantitativa del mapeo en la memoria de una computadora personal (PC).

Posteriormente con los postulados se obtendrá la ecuación matemática del mapeo, cuyas raíces algebraicas permitirán establecer el procedimiento para demostrar el mapeo bajo la perspectiva del teorema de Cantor.

Finalmente, el procedimiento se extrapolará a una estructura lógica que se diseñará con los principios básicos del diseño de sistemas digitales. Esto podría contribuir a la formación del programa de Doctorado de Ingeniería de Cómputo que el Centro de Investigación en Computación tiene como uno de sus objetivos.

Análisis de Modelos de Componentes

Dr. Alfonso Miguel Reyes
Instituto Mexicano del Petróleo, CIC-IPN.
Lic. Elizabeth Acosta Gonzaga
Investigadora del CIDETEC-IPN.

Este artículo describe un análisis de dos de los Modelos de Componentes más usados en la industria del software: COM (Component Object Model) y EJB (Enterprise Java Beans), con la finalidad de obtener una estructura general de un Modelo de Componentes, sus elementos, características y procesos comunes, que permitan establecer un marco de referencia a futuros trabajos de modelos de componentes, así como abrir líneas de investigación sobre estos elementos que coadyuven a una mejor estandarización e interoperabilidad entre modelos diferentes.

Componentes de Software

Un componente de software es definido como [MLUM99, p. 8]:

"Un componente de software es una abstracción estática con enchufes, parte de una estructura, con una arquitectura de software que determina las interfaces que los componentes pueden tener y las reglas que gobiernan su composición".

La característica estática se debe a la instanciación que sufre para ser usado. El componente tiene enchufes, los cuales son usados para proporcionar servicios, pero también para requerir de ellos. Los enchufes son el principal prerrequisito para composición. Un enchufe es una interfaz, pero qué tipo es exactamente ésta, y cómo se enchufan conjuntamente, depende de una estructura de componentes a otra.

Un componente no es usado en forma aislada, sino en el contexto de una arquitectura de software que determina cómo los componentes son enchufados conjuntamente. La estructura, la arquitectura de software y las reglas de composición pueden variar entre los diferentes modelos de componentes.

Szypersky [CSZY97, p. xiii] define a un componente como:

"Una unidad binaria de producción, adquisición e instalación independiente, que interactúa con otros para formar un sistema funcionando".

Un componente de software finalmente es la implementación de un modelo conceptual, producto de una descomposición funcional realizado por el arquitecto de un sistema para obtener la estructura que satisfaga una aplicación de software [PHOS00, p. 429].

Modelos de Componentes.

Recientemente han emergido Modelos de Componentes como una manera para construir aplicaciones fácil y rápidamente, creando una simple estructura para combinar, reusar y transformar componentes de software [RHTS98, p.127].

Las características generales de un modelo de componentes pueden ser definidas como [PHOS00, p. 7]:

1. Un componente es una construcción autocontenida que tiene un uso definido, tiene una interfaz en tiempo de ejecución, puede ser instalado automáticamente y es construido con el conocimiento de un código pegamento específico.

2. El código pegamento es un software que proporciona una bien definida y bien conocida interfaz en tiempo de ejecución para soportar una infraestructura en la cual el componente se fijará. Una interfaz en tiempo de diseño sólo es necesaria pero no suficiente porque ésta no existe en tiempo de ejecución, a menos que sea implementada por alguna pieza de software, esto es, la infraestructura.

3. Un componente es construido por composición y colaboración con otros componentes.

4. Un código pegamento y los correspondientes componentes son diseñados para uso de una persona con conjunto definido de habilidades y herramientas.

Un modelo de Componentes es definido como:

Una estructura metodológica para desarrollar aplicaciones de software usando componentes de software, en el que se definen los atributos, características, mecanismos, herramientas y procesos empleados para lograr la composición de los diferentes elementos de software denominados componentes.

Un atributo es una propiedad del componente, la cual conserva indistintamente y es parte de la definición del modelo. Los atributos constituyen los elementos que definen al modelo.

Las características son el conjunto de cualidades que distinguen a un modelo de componentes de otro y se obtienen en función del ambiente que define el componente, de sus atributos y capacidades.

Los mecanismos de composición definen la forma de realizar la composición de los diferentes elementos del componente.

Las herramientas de composición definen los mecanismos que permiten extender las habilidades del desarrollador, para lograr la creación de componentes.

Un proceso define la forma cómo es creado y usado un componente, lo que impone un estilo de desarrollo e implementación de aplicaciones.

Cada modelo de componente es implementado a través de uno o varios frameworks (estructuras), los cuales constituyen las plataformas de desarrollo de aplicaciones con componentes. Éstos incorporan los mecanismos necesarios para lograr su distribución y composición a través de la red. Por lo que un Modelo de Componentes es un framework horizontal y vertical al mismo tiempo, y también es una herramienta que facilita el desarrollo de aplicaciones distribuidas.

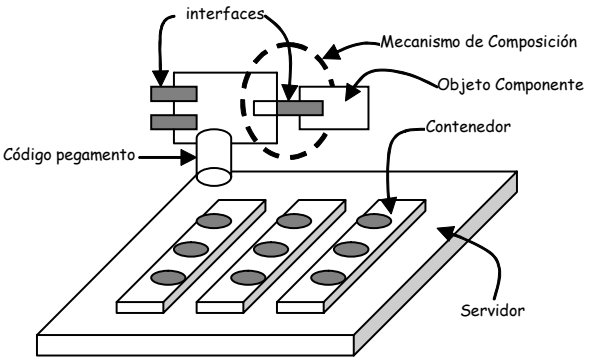

Figura 1. Diagrama de un Modelo de Componentes.

La **figura 1** muestra el diagrama de un Modelo de Componentes.

Atributos de un Modelo de Componentes

Un atributo es una propiedad del componente. Los atributos constituyen parte de los elementos que definen al modelo y son inherentes a éste.

Nombres

Un nombre indica un recurso del sistema, el cual puede ser un dato, componente, evento, etc., cuyas características son significativas, relevantes y asociadas en un contexto específico. Un Modelo de Componentes debe resolver la forma de nombrar los diferentes recursos de aplicaciones en un contexto distribuido.

Interfaces

Una interfaz es un atributo que permite definir puntos de conexión en un componente, para acoplar diferentes elementos. Ésta define una frontera o límite entre dos componentes que permite intercambiar información o interoperar. Szypersky proporciona la siguiente definición [CSZY97, p.174]:

"*Una colección de definiciones de operaciones, cada una con una firma y posiblemente un tipo de retorno. La firma de la operación define el número, tipos y modos de paso de parámetros*".

Servidor/Contenedor

Un servidor/contenedor proporciona un contexto de desarrollo, instalación, ejecución y ciclo de vida de los componentes. De desarrollo, porque ofrece mecanismos para que los componentes puedan interactuar entre ellos; de ejecución, al proporcionar mecanismos que permiten instanciar los componentes requeridos por las aplicaciones en ejecución; de instalación porque ofrece los mecanismos necesarios para registrar los nuevos componentes y de ciclo de vida, al permitir llevar la administración de la creación, uso y destrucción de componentes.

El Servidor implementa los servicios de más bajo nivel del modelo de componentes, como son aquellos asociados con el uso de servicios de las plataformas de middleware. El contenedor implementa los servicios de más alto nivel, como son los asociados con la administración y ciclo de vida de los componentes.

Fábricas de componentes

Un atributo distintivo de un modelo de componentes es la forma como implementa la instanciación de sus componentes, ya que no existe un

mecanismo como el *"new"* de los lenguajes de programación orientados a objetos; éstos deben implementar lo que se denomina "fábricas de componentes", la cual define un objeto que permite crear otro objeto [RHIG96, p.126].

Código Pegamento o Enchufe

El código pegamento o enchufe define la forma cómo un componente es enchufado o pegado al servidor/contenedor, lo que permite que pueda ser utilizado y enlazado por otros componentes. Cada modelo de componentes define una forma particular de realizarlo.

Objeto Componente

Un objeto componente o componente constituye la definición de la unidad mínima de instalación, administración e instanciación en un modelo de componentes, la cual es registrada por el Servidor/Contenedor y corresponde básicamente a la definición de la interfaz del componente, la que a su vez define una unidad de instanciación.

Características de un Modelo de Componentes.

Un modelo de componentes puede tener varias características [MLUM99, p. 9], aunque el número de éstas puede ser tan grande como el número de modelos de componentes existentes. Sólo se analizan los más generales.

1. Escala y Granularidad.- Define el tamaño del componente.

2. Código Binario o Fuente.- Indica la forma final del código del componente.

3. Homogeneidad o Heterogeneidad.- Indica la aceptación en la composición de elementos de otros modelos en forma directa, sin intervención por parte del desarrollador en el diseño de adaptadores o transformadores.

4. Caja Blanca o Caja Negra.- Manifiesta la forma ó capacidad como el componente es exportado.

5. Con Estado o sin Estado.- Indica la posibilidad de modificar los estados de los componentes, sobre todo los referentes a los atributos del componente sin necesidad de codificación.

6. Metacomponentes.- Indica la capacidad del componente para describir su composición, y los elementos que la integran en forma dinámica en tiempo de ejecución, por lo que se dice es reflectivo. Un metacomponente define un componente con capacidades reflectivas.

7. Administración de Versiones.-Indica los mecanismos que permitan identificar y hacer compatibles diferentes versiones de un mismo componente.

8. Tipos.- Identifica mecanismos de uso ó verificación de tipos. De interés especial es la consideración que un tipo es una interfaz con un contrato simplificado [CSZY97, p. 77].

9. Distribución.- Es la capacidad del modelo con mecanismos necesarios para lograr su distribución y composición a través de la red.

El Modelo de Objetos de Componentes (COM, Component Object Model)

COM es resultado del proceso evolutivo de tecnología de Microsoft, ya que unifica diferentes criterios utilizados en diferentes productos: bibliotecas DLL (Dynamic Linking and Loading), DDE (Dynamic Data Exchange), objetos OLE (Object Linking and Embedding), objetos OLE2, etc. [DROG97, p. 12].

Un componente es una abstracción con enchufes, los cuales le permiten agregarse o incorporarse a otros, utilizando algún mecanismo de composición. COM especifica la arquitectura de un Modelo de Componentes que permite la composición de elementos denominados objetos componentes. Los enchufes son denominados interfaces, las cuales definen los puntos de conexión entre diferentes elementos de un componente, además son el medio de composición de los elementos.

Los objetos componentes son los elementos básicos de composición, éstos representan unidades de implementación de los servicios ofrecidos por el componente. Un objeto componente es construido para satisfacer un requerimiento del sistema o aplicación. Un requerimiento de un sistema puede ser tan simple que un solo componente sea necesario para su satisfacción o tan general que es dividido en diferentes subrequerimientos, lo que obliga a los diseñadores a desarrollar diferentes niveles de abstracción. Cada nivel de abstracción puede ser representado por uno o varios componentes, los cuales a su vez son utilizados como elementos de composición posteriormente.

En su definición más sencilla un servidor es un proceso proveedor de servicios que son demandados por un conjunto de procesos denominados clientes. En COM un servidor es un mecanismo de agrupación de componentes.

La **figura 2** muestra el diagrama del Modelo de Componentes COM.

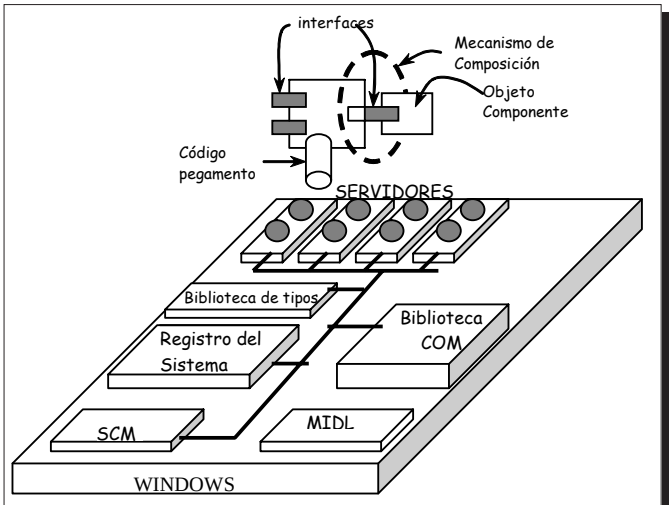

Figura 2. Modelo de componentes COM.

ATRIBUTOS

INTERFACES

Una interfaz proporciona una conexión entre dos diferentes objetos, ya que permite que un objeto use una función implementada en otro objeto, por lo que la interfaz los encadena, como se muestra en la **Figura 3**.

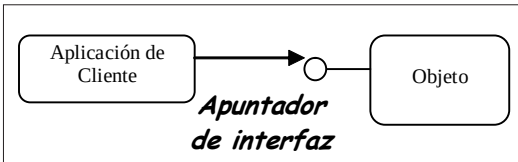

Figura 3. Interfaz.

Una de las características del modelo de componentes COM es la independencia del lenguaje de programación, por la utilización de un Lenguaje de Definición de Interfaces Microsoft (MIDL por sus siglas en ingles), el cual permite definir interfaces que son independientes de un lenguaje de programación, con la estructura mostrada en la **Figura 4**.

La interfaz debe tener al menos dos atributos: ***Object*** para indicar que es una interfaz COM y ***el nombre físico de la interfaz, e****l cual es representado por un Identificador Global Unico (GUID por sus siglas en Inglés) de 128 bits*, obtenido por medio de la función ***HRESULT CoCreateGuid (GUID *pguid).***

INTERFAZ IUNKNOWN

Las interfaces usadas en COM deben ser derivadas de la ***interfaz Iunknown***, la cual es definida por el archivo UNKNOWN.H o UNKNOWN.IDL. Ésta constituye la base de composición de los componentes COM, ya que contiene tres funciones miembros que le permiten navegar a través de los componentes y manejar su ciclo de vida: ***QueryInterface(), AddRef(), Release().***

La **figura 5** muestra la definición de la interfaz IUnknown.

```
Interface IUnknown
{
    virtual HRESULT STDMETHODIMP QueryIntrface(REFIID riid, void *ppv) = 0;
    virtual ULONG STDMETHODIMP AddRef(void) = 0;
    virtual ULONG STDMETHODIMP Release(void) = 0;
}
```

Figura 5. Interfaz Unknown.

MÉTODO QUERYINTERFACE

La interfaz IQueryInterface comprueba que el componente y el proceso que lo llama son realmente compatibles. *QueryInterface* es la parte más importante de COM ya que las interfaces que un componente soporta son las que *QueryInterface* regresa un apuntador de interfaz.

MÉTODOS ADDREF() Y RELEASE()

AddRef y Release implementan una técnica de administración de memoria llamada ***conteo de referencias***. Un componente COM mantiene un número llamado cuenta de referencia. Cuando un cliente toma una interfaz desde el componente, la cuenta de referencias es incrementada y cuando el cliente finaliza el uso de la interfaz, la cuenta de referencia es decrementada. Cuando la cuenta de referencias llega a cero, la instancia de la clase es borrada.

```
[ atributo1, atributo2,.....,atributon, ..]
interface IEstaInterfaz : IInterfazBase
{
        typedef1;
        typedef2;
        .
        .
        method1;
        method2;
}
```

Figura 4. Estructura de un MIDL.

NOMBRES

COM no define un servicio de nombres para administración de contextos de nombres, los cuales permiten resolver los problemas asociados con la identificación de nombres en un contexto distribuido. En este caso ofrece algoritmos para la generación de identificadores únicos (GUID), los cuales son asociados a nombres de interfaces, los que a su vez identifican a un componente.

Los GUID's residen en el registro del sistema (system registry), la cual puede ser explorada con el programa ***regedit.***

SERVIDORES

Un servidor COM es un archivo binario que agrupa el código de métodos para uno o más objetos COM.

Un servidor puede ser empacado como una librería de enlace dinámico o un archivo ejecutable normal. Cada máquina que soporta COM tiene un SCM (Service Control Manager), que es el encargado de los servicios de activación de los objetos COM (servidor) y encadenar el apuntador de interfaz inicial (IUnknown) [DBOX98, p.100-114]. Cada objeto COM corre dentro de un servidor. Un solo servidor puede soportar múltiples objetos COM como se muestra en la **figura 6**, existiendo tres formas como un cliente puede accesar a objetos COM proveídos por un servidor: Servidor en proceso, Proxy de objeto local y Proxy de objeto remoto.

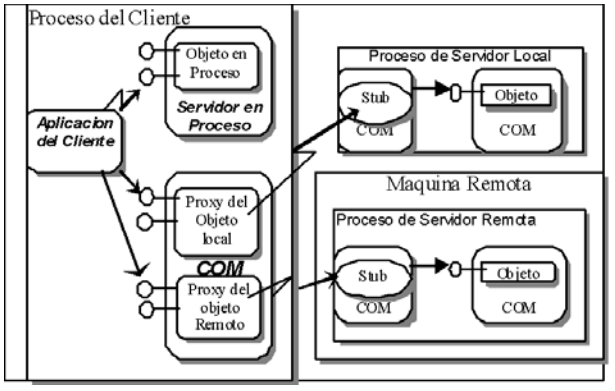

Figura 6. Servidor y objetos COM.

Código Pegamento o Enchufe

Este tipo de código se encuentra compuesto por una clase objeto, los puntos de entrada del servidor y las funciones de registro en el sistema.

Fábricas de Clases

Existen dos interfaces que pueden ser usadas como fábricas de clases, *IClassFactory* e *IclassFactory2*. Éstas tienen dos métodos: *CreateInstance* y *LockServer,* las cuales permiten crear una instancia de una clase COM. LockServer evita la caída o descarga de un servidor cuando este aún puede ser utilizado.

Objeto Componente

Un objeto COM no es un objeto en el sentido estricto de orientación a objetos, las interfaces COM no tienen estado y no pueden ser instanciadas para crear un objeto único. Una interfaz es simplemente un grupo de funciones relacionadas, los clientes de COM obtienen un apuntador para acceder a las funciones de una interfaz.

Una sola interfaz define a un objeto componente, aunque puede soportar cualquier número de interfaces, como se muestra en la **figura 7**.

Bibliotecas de Tipos

Una biblioteca de tipos define un archivo binario de encabezados independiente de los lenguajes de programación. Éste es creado por el compilador MIDL a partir de la definición del archivo IDL del componente. Este archivo contiene los nombres de las clases e interfaces que son implementadas con el servidor, número y tipo de los parámetros para cada método, así como los GUID's para cada clase e interfaces.

Mecanismos de Composición

La forma de reutilización en COM es a partir de la definición de nuevas interfaces que en esencia crean nuevos componentes COM, las cuales pueden extender interfaces existentes, agregando nueva funcionalidad requerida. COM soporta dos mecanismos de composición: la contención o inclusión y la agregación de interfaces. En ambos casos el componente externo controla el ciclo de vida de los componentes internos e IUnknown representa las interfaces de los componentes internos.

La contención/delegación/inclusión en COM permite que un componente externo delegue o reenvíe las invocaciones de funciones o métodos ofrecidas por el componente, las cuales se encuentran en componentes internos. La **Figura 8** muestra un ejemplo de este tipo de mecanismo.

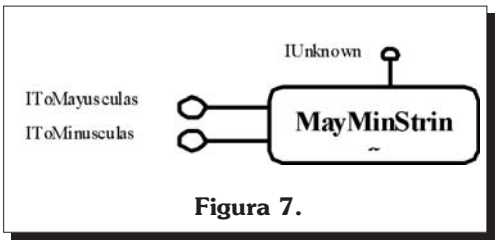

Figura 7.

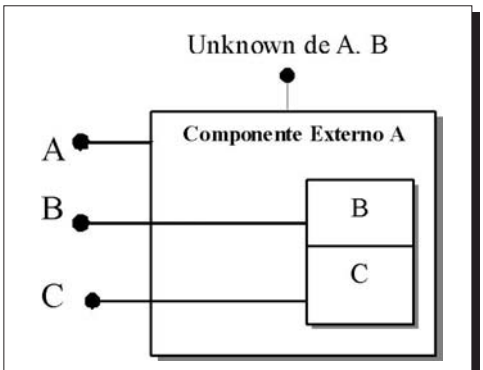

Figura 8. Contención/delegación/ inclusión.

En la agregación en lugar de reenviar cada llamada, el componente externo *IUnknown* expone directamente los apuntadores de las interfaces de los componentes internos a sus clientes, es decir el componente externo expone las interfaces de los componentes internos como suyas,

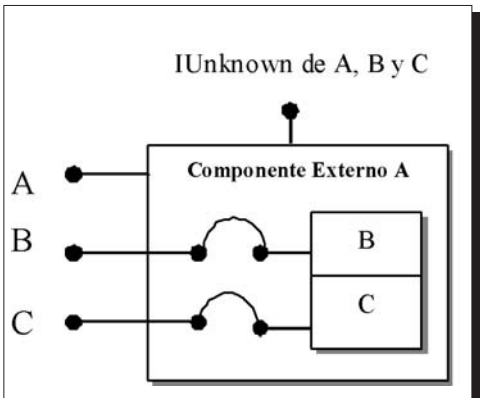

Figura 9.

como se muestra en la **Figura 9**.

Procesos

Los procesos básicos del modelo COM son: Creación y Uso de componentes COM. El proceso de creación de un componente COM es el siguiente:

1. Crear un proyecto DLL nuevo.
2. Crear un archivo de interfaz usando IDL.
3. Declarar una clase C++ que implemente la interfaz COM.
4. Implementar las interfaces declaradas.
5. Crear una clase objeto.
6. Crear los puntos de entrada requeridos por el DLL.
7. Crear las funciones de registro en el sistema.
8. Crear el cliente.

La creación del Cliente realiza los siguientes pasos:

1. Iniciando / Cargando un Servidor.
2. Instanciando un Componente.
3. Uso del apuntador de la interfaz del componente o referencia a éste.

Características

La siguiente tabla resume las características del modelo de componentes COM.

Característica		Comentarios
1	Escala y Granularidad	De granularidad pequeña a grande, a través de composición de interfaces.
2	Código Binario o Fuente	Código Binario
3 4	Homogeneidad o Heterogeneidad	Los componentes son heterogéneos en el lenguaje de programación, y en plataformas de hardware solo aquellas compatibles con Windows. Existe composición con otros modelos de componentes a través de técnicas de interoperabilidad, no en forma directa.
5	Caja Blanca o Caja Negra	Caja negra
6	Con Estado o sin Estado	Sin Estado.
7	Meta componentes	No
8	Administración de Versiones	No
9	Tipos	Si
10	Distribución	Si, con el uso de DCOM

JavaBeans Enterprise

JavaBeans Enterprise define un Modelo de Componentes como una arquitectura para el desarrollo y la implementación de aplicaciones distribuidas de negocios basados en componentes [SMIC99, p.15]. La arquitectura define como elemento base lo que denomina un bean empresarial, equivalente a un componente. Para entender el concepto de un componente EJB (Enterprise Java Beans), es necesario comprender el modelo de proceso de desarrollo de aplicaciones utilizando EJB. Este modelo supone cuatro fases de desarrollo:

1. Desarrollo de componentes EJB conteniendo la lógica del negocio.
2. Desarrollo de servidores/contenedores de aplicaciones para uso de plataformas de instalación y acoplamiento de componentes EJB.
3. Desarrollo de aplicaciones integrando y acoplando diversos componentes EJB en un servidor/contenedor.
4. Instalación y mantenimiento de las aplicaciones.

El proceso de desarrollo reconoce seis papeles o responsabilidades que adoptan los diseñadores de las aplicaciones, con los siguientes roles:

1. Proveedor de Beans.
2. El Proveedor de Contene dores.
3. El Proveedor de Servido res.
4. El Ensamblador de Aplica ciones.
5. El Instalador.
6. El Administrador del Siste ma.

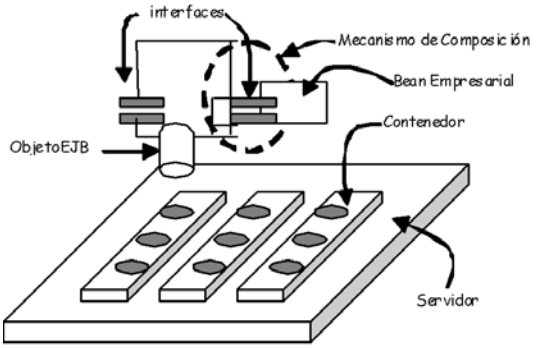

Figura 10. Diagrama del Modelo de Componentes EJB.

La **figura 10** muestra un diagrama del Modelo de Componentes EJB.

Atributos

Nombres

La forma de dar solución al problema de nombres en el modelo de componentes EJB, depende del proveedor del servidor/contenedor, ya que éste define el tipo de servicios de middleware que son soportados. Dos modelos son soportados ampliamente: El servicio de Nombres de CORBA (Common Object Request Broker Architecture) y el servicio de nombres de JNDI (Java Naming and Directory Interface).

JNDI es un sistema para clientes Java basado en acceso a estructuras de directorios de diferentes proveedores de sistemas distribuidos (LDAP (Lightweight Directory Acces Protocol), NIS (Network Information Services), SLP (Service Location Protocol), CORBA Name Services, etc.).

El Servicio de Nombres de CORBA permite formar contextos de nombres en una estructura jerárquica de árbol, la cual permite la navegación una vez obtenido un contexto inicial.

Interfaces

EJB usa el modelo de interfaces usadas en el lenguaje de programación Java, por lo que todos los componentes EJB son derivados de clases especiales.

La clase Bean Empresarial constituye el modelo de implementación del bean base definida a través de una interfaz, la cual proporciona los enchufes por los que el componente se incorpora a un contenedor y se conecta a los clientes [EROM, p.72]. Las clases bean empresariales son derivadas a partir de la interfaz *javax.ejb. EnterpriseBean,* además incluye otros dos tipos de interfaces: Remota y Hogar

INTERFAZ REMOTA

La Interfaz Remota conjunta todos los métodos que el componente expone a los clientes. Ésta se deriva de la interfaz *javax.ejb.EJBObject,* y se implementa por un objeto Hogar como se muestra en la Figura 11, posee un conjunto de métodos que permite la administración de los elementos del componente.

INTERFAZ HOGAR

La interfaz Hogar proporciona la definición de los métodos para crear, encontrar y remover los elementos del componente, se deriva de la interfaz *javax.ejb.EJBHome,* la cual es implementada por un objeto EJB como se muestra en la **Figura 11**.

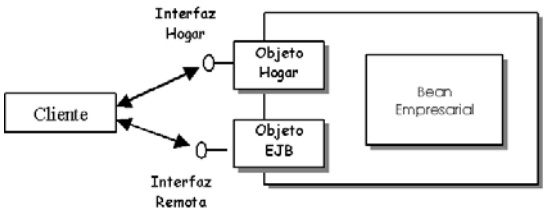

Figura 11. Interfaz Hogar, derivada de la interfaz *javax.ejb.EJBHome*.

CONTENEDOR/SERVIDOR EJB

El concepto de contenedor y servidor en EJB no es claramente separado [EROM99, p. 46], ya que el modelo EJB los define como una sola entidad. Un servidor EJB proporciona un ambiente para ejecución de uno ó más contenedores, éste administra los recursos de bajo nivel del sistema, asignándoselos a los contenedores como son requeridos. El contenedor proporciona una base donde los beans pueden ejecutarse; es responsable de administrar los beans que están corriendo en él. Los contenedores son responsables de conectar los clientes con los beans, realizar la coordinación de las transacciones, proveer persistencia, administrar el ciclo de vida, etc. La **figura 12** muestra el diagrama de un Servidor/Contenedor EJB.

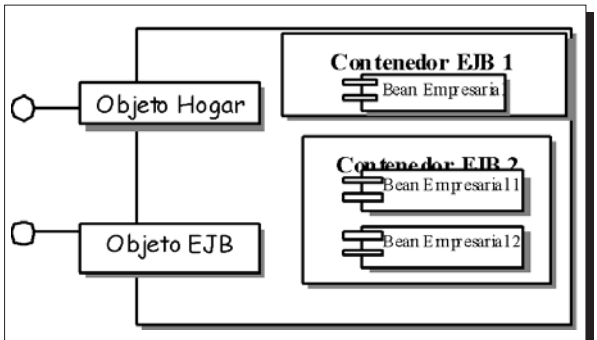

Figura 12. Diagrama de un Servidor/Contenedor EJB.

Un Servidor/Contenedor tiene la responsabilidad de administrar: las transacciones distribuidas, la seguridad, los recursos y ciclo de vida, la persistencia, la accesibilidad remota, el soporte multicliente y localizar en forma transparente nombres.

CONTEXTOS

Un contenedor almacena información relacionada con la operación de los beans en un objeto llamado ***objeto contexto EJB.*** Un objeto contexto representa un camino por el cual los beans pueden realizar llamadas de regreso al contenedor, para conocer su estado actual y modificarlo si es necesario. La información guardada en los contextos pertenece a los objetos hogar, objetos EJB, interacciones del bean, seguridad, y propiedades ambientales de la instalación del bean. Los beans poseen también un contexto propio.

CÓDIGO PEGAMENTO: OBJETOS EJB

Los clientes no pueden instanciar directamente un componente, ya que la invocación es atrapada por el contenedor y ***delegada*** a la instancia del bean. Un componente no puede ser llamado directamente a través de la red; ya que no posee habilidades de red, en este caso el contenedor realiza el trabajo correspondiente, además de guardar toda la información de uso de los beans contenidos, fungiendo como una capa de indirección entre el cliente y el bean. Esta capa de indirección es lo que se denomina ***Objeto EJB*** y es un representante que conoce toda la operación del componente, sirviendo como pegamento (GLUE) entre el cliente y el componente, como se muestra en la **Figura 13**.

FÁBRICAS DE CLASES: OBJETOS HOGAR.

Un cliente no puede instanciar un bean directamente, entonces: ¿Cómo adquiere referencias a los elementos

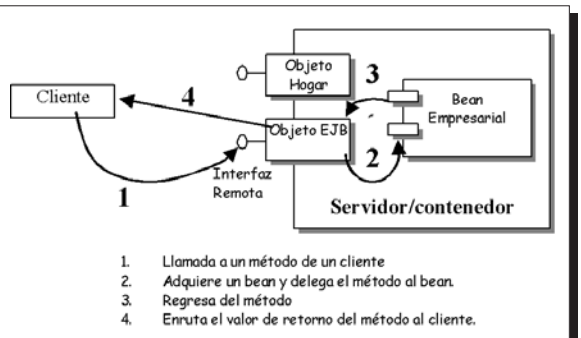

Figura 13. Capa de indirección u *Objeto EJB*.

del componente y cómo los destruye?. El mecanismo empleado es usando fabricas de clases, las cuales permiten la instanciación de objetos. Estas fabricas de clases en EJB son denominadas Objetos Hogar y tienen como responsabilidades: Crear, Localizar y Remover elementos.

Objeto Componente: Bean Empresarial (BE).

En el desarrollo de una aplicación se distinguen dos tipos de elementos que realizan diferentes tipos de pruebas, uno está orientado al desarrollo de transacciones cuyo objetivo es el procesamiento de información en general, y es llamado *componente de lógica de aplicación*; el otro se encuentra orientado a operaciones relacionados con el almacenamiento o manejo de datos por parte de la aplicación, *denominado componente de datos persistentes.* Un Bean Empresarial de Sesión representa un componente de lógica de aplicación, éste puede ser con estado o sin estado. Los beans de sesión con estado tienen la capacidad de registrar la información de los clientes, con los que están interactuando. Los beans de sesión sin estado no registran información de los clientes con los que interactuan.

Un Bean Empresarial de Entidad representa un componente de datos persistentes, usado para modelar datos. *Una instancia de un bean de entidad es la vista en memoria de la base de datos.* Los datos de un bean de entidad (o instancia de datos) son el conjunto físico de datos, almacenados en una base de datos.

Un bean de entidad es de larga duración, que sobrevive a las fallas, requiere la implementación de dos operaciones por parte del bean: *ejbLoad() y ejbStore()* para uso con una Base de Datos; *ejbActivate() y ejbPassivate(),* para optimizar el uso de recursos durante el apilamiento de operaciones; *ejbCreate(), ejbRemove(), y ejbFind*()* para administración de recursos del bean y ciclo de vida.

Un Bean Empresarial de Sesión ó Entidad se encuentra compuesto básicamente de los siguientes elementos:

1. Clase bean Empresarial.
2. Interfaz Remota.
3. Objeto EJB.
4. Interfaz Hogar.
5. Objeto Hogar.
6. Descriptor de Instalación.
7. Manifiesto.
8. Archivo Ejb-jar.

Un Bean Empresarial de Entidad agrega una Clase Llave Primaria, que es el identificador único del bean.

Los elementos 1, 2, 3, 4, 5 ya fueron descritos.

Descriptor de Implementación

Es un archivo que permite describir los requerimientos de servicios de middleware del componente.

Propiedades Específicas del Bean

Un archivo de propiedades específicas del componente puede ser incluido. Éstas son leídas en tiempo de ejecución para afinar las funciones del bean.

Archivo Ejb-jar

El archivo Ejb-jar constituye la última fase del proceso de desarrollo de un componente EJB, ya que es la fase de empacamiento del componente en un formato .JAR

Mecanismos de Composición

El mecanismo básico de composición de EJB es el de contención.

Procesos

El procedimiento general de construcción de bean de sesión sin estado es el siguiente:

1. Declaración de la interfaz remota
2. Implementación del bean Empresarial
3. Construcción de la interfaz Hogar
4. Escribir el Descriptor de Instalación.
5. Construir el archivo EJB-jar.
6. Escribir el código del cliente.

Un bean de sesión con estado requiere de mayores recursos, ya que es necesario guardar la información generada con la interacción con cada uno de sus clientes. Como la cantidad de clientes realizando peticiones a un bean con estado puede ser mayor a la capacidad disponible del bean, es necesario un mecanismo que permita maximizar el aprovechamiento de sus recursos, extendiéndolos de una manera virtual. El bean usa ***ejbPassivation()***, que permite liberar recursos al contenedor al almacenar en dispositivos secundarios la conversación del bean, y ***ejbactivation()***, que permite recuperar la conversación de un bean de un almacén secundario, como se muestra en la **figura 14**.

El procedimiento para escribir un bean de entidad es similar el de sesión, lo que cambia son los paráme-

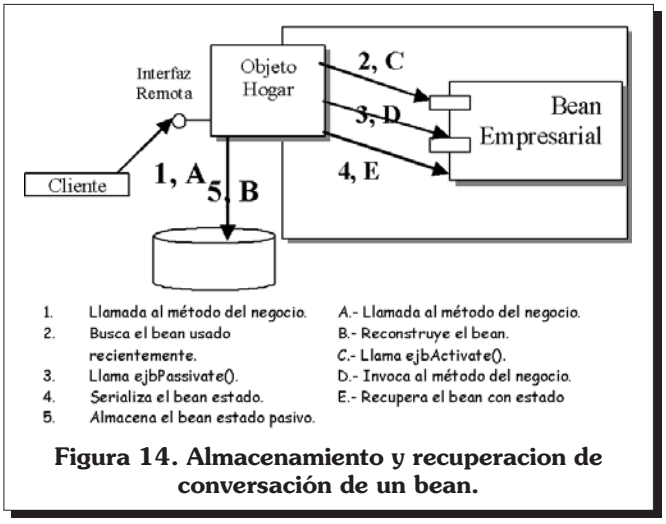

Figura 14. Almacenamiento y recuperacion de conversación de un bean.

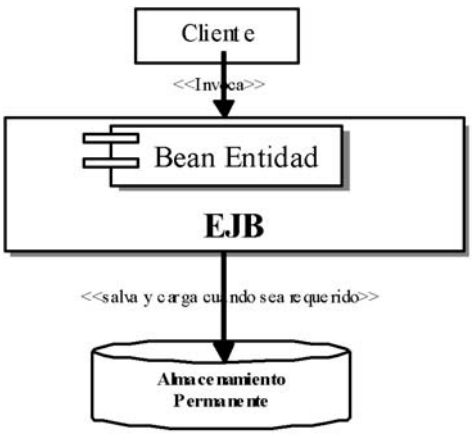

Figura 15. Interacción de un cliente con el Bean y el almacenamiento persistente.

tros que deben utilizarse en la herramienta de instalación para dotar de persistencia al bean, generalmente una base de datos, la cual debe estar trabajando. La **figura 15** muestra la forma como un cliente interactúa con el Bean y el almacenamiento persistente.

Un Bean Entidad a diferencia de un bean de sesión puede ser compartido por varios clientes. Además que un bean entidad debe modelar las características de transaccionalidad, concurrencia y recuperación, requeridas por unidades de información.

Una característica importante del modelo EJB es el Código Pegamento, el cual es generado en forma automática por el Servidor/Contenedor durante la instalación del Bean Empresarial, a partir de la información de la interfaz remota y hogar. Ésta es la razón por la que no aparece en el procedimiento de construcción del bean.

Característica	Comentarios
1. Escala y Granularidad	De granularidad pequeña a grande, a través de composición Beans Empresariales
2. Código Binario o Fuente	Código Bytecode
3. Homogeneidad o Heterogeneidad	Los componentes son homogéneos en el lenguaje de programación, y heterogéneos en plataformas de hardware. Existe composición con otros modelos de componentes a través de técnicas de interoperabilidad, no en forma directa.
4. Caja Blanca, Negra ó Gris	Caja Gris
5. Con Estado o sin Estado	Con Estado.
6. Meta componentes	Si
7. Administración de Versiones	No
8. Tipos	Sí
9. Distribución	Si, usando RMI ó CORBA

Características

La tabla anterior resume las características del modelo de componentes EJB.

Conclusiones

El análisis de los Modelos de Componentes construyó un marco de referencia para:

1. Una estructura común.
2. Identificar elementos comunes de un Modelo de Componentes.- En los siguientes años surgirán Modelos de Componentes con diferentes características, por lo que es necesario tener un marco de referencia para identificar los elementos comunes de análisis.
3. Comparación de Modelos de Componentes.
4. Facilitar toma de decisiones acerca de la selección de un Modelo de Componentes adecuado a dominios de aplicaciones especificas.
5. Abrir líneas de investigación en elementos comunes de los modelos, que faciliten la estandarización e interoperabilidad entre diferentes modelos.
6. COM y EJB son dos modelos de componentes con elementos comunes de diseño e implementación diferente. Ambos carecen de un Servicio de Eventos.

Bibliografía

CSZY97 Clemens Szyperski; Component Software, Beyond Object-Oriented Programming. Editorial Addison Wesley, 1997.

MLUM99 Marcus Lumpe. A pi-Calculus Based Approach for Software Composition. Bern University Ph. D. Thesis, 1999.

DBOX98 Don Box. Essential COM. Editorial Addison Wesley, 1998.

EROM99 Ed Roman. Mastering Enterprise JavaBeans. Editorial Wiley, 1999.

SMIC99 Sun Microsystems. Enterprise JavaBeans, Specification, v1.1. 24-Nov-1999.

DROG97 Dale Rogerson. Inside COM. Editorial Microsoft Press, 1997.

AGOR00 Alan Gordon. The COM and COM+, Programming Primer. Microsoft Technologies Series. Editorial Prentice Hall, 2000.

RHTS98 Reaz Hoque and Tarun Sharma. Programming Web Components. Editorial McGraw Hill, 1998.

PHOS00 Peter Herzum and Oliver Sims. Business Component Factory. John Wiley and Sons Inc., 2000.

RHIG96 Robert H. High Jr. Component Model for Managed Objects in Large-Scale Distributed

CUC'96 Component-Based Software Engineering. Editorial Cambridge University Press, 1998.

Agradecimientos:
Al Instituto Mexicano del Petróleo por las facilidades brindadas para este trabajo, en especial al Ing. Tomás Ramírez Maldonado.

La Robótica, Principio y Evolución

M. en C. Francisco F. Córdova Quiroz
Profesor de la UVM

Para muchas personas, la palabra robótica simboliza el *súmmum* de la tecnología, esto es, un mundo separado de la vida real. En la práctica, esta herramienta de trabajo, resultado de la unión de la ingeniería aplicada en varias áreas y con poco más de un siglo de existencia, facilita el trabajo en muchas y muy variadas actividades cotidianas.

Lo mismo sucede con el concepto de robot industrial, erróneamente asociado a las ideas de destrucción, desplazamiento y pérdida de fuentes de trabajo. Así, se debe analizar la evolución de los robots en relación a la historia humana, para estudiar desde una posición objetiva el empleo, los beneficios y los perjuicios generados por los robots sobre nuestras sociedades.

Evolución

El término "robot" se deriva de la palabra checoslovaca *Robota*, que significa *trabajo forzado* **[1]**, y se acuñó por el escritor checo Karl Capek en su obra "R.U.R.", de 1920. En ella, la trama gira en torno a maquinas similares al hombre, capaces de realizar trabajos difíciles y peligrosos, y que al final terminan por dominar al ser humano.

En la obra, la idea esencial es fabricar robots que sirvan al genero humano, facilitando el trabajo y liberandolo de las actividades peligrosas. La idea realmente no era tan novedosa como se consideró en 1920, ya que antes gente como Leonardo da Vinci habían pensado en este tipo de maquinas. De igual forma en la antigua Grecia los filósofos ya veían la posibilidad de crear artefactos (artilugios) para realizar actividades de limpieza y cultivo dentro de sus fincas, dandoles el nombre de Humanoides, ya que tendrían forma humana.

Posteriormente, personalidades como Newton y Descartes, vislumbraban una máquina o medio por el cual el hombre se liberara de las actividades monótonas y rutinarias; particularmente se referían a la solución de problemas matemáticos, ya que consideraban que: *El hombre es un ser universal y creativo, el cual no debe esclavizarse en la solución metódica y repetitiva de problemas*. Consecuentemente, con la aparición de esta maquina el hombre mismo seria desencadenado y liberado de tales actividades, para así utilizar mejor su potencial intelectual y creativo.

Los robots han tenido una evolución constante desde su creación, y aún cuando no se conocían como tales, este tipo de aparatos se puede ubicar históricamente a partir del siglo XVI, en la Corte de Luis XV en Francia. Inicialmente, surgieron por un accidente trágico, debido a la muerte de la familia de un relojero, quién decidió remplazarla por muñecos mecánicos. Estos estaban constituidos con base en un complicado sistema de engranes sincronizados, tal como en un reloj.

Dada la novedad que despertaron estas máquinas, el Rey mando fabricar varias de ellas para su entretenimiento. Así, los primeros robots tienen funciones meramente ornamentales y de diversión; después de este hecho, la siguiente referencia histórica a mecanismos de este tipo se dá durante la revolución industrial.

Partiendo del uso de engranes y levas, y con la inclusión de la maquina de vapor, fue posible iniciar la automatización de los procesos productivos, siendo aquí donde se puede definir por primera vez a un robot bajo una concepción netamente industrial:

Un robot es una máquina que puede realizar una serie de actividades repetitivas sin la necesidad de la supervisión humana.

No es de extrañarse que esta primera definición se asemeje a la de la automatización de un sistema, ya que en realidad los robots de esa época se construyerón con un propósito productivo y lucrativo, enfocado a la fabricación.

En esta etapa histórica se presenta un fenómeno social muy fuerte y plenamente documentado, referente

a la pérdida masiva de trabajos y fuentes de empleo en las factorías, principalmente en la industria textil. Al implantar estos robots primitivos en la producción, con funciones plenamente definidas, su rápidez y eficiencia probocaron el desplazamiento de los obreros como fuerza de trabajo, causando con esto el descontento social general.

Sin embargo, el uso constante de estos aparatos y el inexistente mantenimiento a los mismos, empezó a generar fallas, dando por consecuencia que la mayoría de las fabricas sufrieran un paro paulatino pero inevitable. El industrial se vió en la necesidad de recontratar a sus antiguos obreros, y así mantener la producción, pero debió considerar la contratación y adiestramiento de personal calificado, capaz de mantener en buen estado a los sistemas.

En esta etapa histórica se observa que el empleo de robots como fuerza de trabajo no es realmente una amenaza al hombre; si bien los robots están diseñados para realizar actividades monótonas y repetitivas, sin errores y con una velocidad mucho mayor a la del ser humano, teniendo así una aparente superioridad laboral, las máquinas no son infalibles, siendo este el punto donde el hombre retoma el control de la situación, ya que sin los seres humanos tarde o temprano las maquinas fallarían. Consecuentemente los robots son dependientes del hombre y no a la inversa, como comúnmente se piensa.

Un efecto colateral del desplazamiento de la fuerza de trabajo fue la busqueda de otras alternativas para la subsistencia, lo cual dió lugar a la creación de nuevos servicios, y en su conjunto, a una evolución y crecimiento tanto de la economía como de la sociedad. A partir de esta etapa, surge el binomio hombre-maquina, presente hasta nuestros dias.

Es hasta el surgimiento de la computación, con el empleo de la computadora como herramienta de trabajo y de control, que se retoma con fuerza el concepto de robot. En la decada de los 60's se da un campo fértil para la creación, y el diseño, diversificandose las posibilidades para el empleo de los robots en la vida diaria. En ese momento se crean robots con un mayor grado de movilidad (grados de libertad), y con mecanismos mas complejos, cuyo tiempo de respuesta (tiempo de acción) es menor; se amplía el rango de potencia y se reduce el consumo de energía.

Se inicia el control de los robots a través de computadoras, siendo el ejemplo mas claro los brazos mecánicos, tantas veces vistos en laboratorios, plantas de ensamble y, como una muestra de modernidad, en muchas películas.

Un brazo mecánico típico responde a impulsos eléctricos codificados; estos son interpretados en el robot por un módulo lógico y transferidos a un módulo de potencia, el cual efectúa el movimiento ordenado. Los impulsos están gobernados por un programa de control, para realizar las acciones y movimientos que la tarea específica requiera. Con todas estas características, se puede redefinir a un robot como:

La unión de sistemas electrónicos y mecánicos, que interactúan entre si para llevar acabo una tarea especifica; dicha tarea se transmite y se controla desde una computadora, mediante un programa. El control se transfiere a través de una interface, que le indica al robot cada una de las acciones y movimientos que debe realizar.

Dada la conjunción de sistemas ya mencionados, un robot dispone tanto de partes electrónicas y mecánicas, como de procedimientos de control y de ajuste para su correcto funcionamiento, siendo sus elementos generales:

1.- Cuerpo del robot
2.- Brazos
3.- Actuadores
4.- Sensores y Transductores
5.- Interfaces
6.- Control de Posición
7.- Control de Potencia
8.- Módulo Lógico

En el caso de los robots controlados por compoutadora evidentemente se requiere un programa para dicho control. Dependiendo del esquema aplicado, el programa puede ejecutarse en una computadora de uso especifico, diseñada para el Robot, desde una PC de propósito general, o bién en un módulo lógico dentro del mismo robot.

La programación de un robot es una acción simple, permitiendo en la mayoría de los casos la redefinición de sus funciones, dentro de los límites establecidos por la configuración mecánica.

Lo anterior ha cambiado el concepto de robot, considerandosele ahora un autómata finito, donde cada instrucción dada por el programa de control es parte de una gramática particular, que indica el estado que debe tomar el autómata; así, una secuencia de estas instrucciones lo lleva a un proceso definido, que en este caso se traduce como un movimiento o acción. Esta característica fue aprovechada por la industria japonesa a finales de los 70's y principios de los 80's, y le permitió ajustar sus líneas de producción y de armado en una forma tal que le hizo ganar gran parte del mercado internacional, por su altos índices de productividad, eficiencia y calidad.

Al ver el éxito industrial y económico de Japón, esta forma de trabajo se adoptó por otros países para mantenerse en el mercado mundial. Incluso en México existen industrias casi completamente automatizadas con líneas de robots, tales como las armadoras automotrices y algunas maquiladoras.

A partir de este punto los Robots (Autómatas), se han extendido en las mas variadas ramas del conocimiento y de las actividades del hombre, tales como la industria, medicina, educación, investigación, diversión, arte, etc.; en muchas de estas actividades los robots han protegido al hombre, haciendo trabajos, exploraciones e investigaciones a distancia sin que este se exponga a condiciones extremas y riesgosas.

Actualidad

Hasta este punto se han descrito algunas situaciones históricas del desarrollo de los Robots, así como sus ventajas y aparentes desventajas, estableciendo a la robótica como una unión no definida entre la mecánica y la electrónica; sin embargo, la robótica sigue evolucionando, y ya en 1969 el Dr. Yaskawa, expandió el termino de robot o de robótica y le dio un rumbo mas amplio, definiendo a esta como parte de la Mecatrónica. La definición generalizada de Mecatrónica es: *La integracion de la Ingeniería Mecanica con la Ingeniería Electronica, la Inteligencia Artificial y el Control Computarizado, para el diseño y manufactura en procesos y productos*.

La inclusión de la Inteligencia Artificial propicia el desarrollo del concepto de Maquinas Inteligentes Concientes o **MIQ**.

Con el surgimiento de la mecatronica se insertan a los robots nuevos elementos, tanto para su control como para su desempeño, independientemente de su ambiente de trabajo. Esto conduce nuevamente a redefinir el término Robot:

Es la Unión Múltiple de Sistemas Electrónicos, Mecánicos, de Control Computacional y de Inteligencia Artificial, los cuales interactúan entre sí en forma compleja, con una mejor disposición de sensores que le permite relacionarse con su medio ambiente para llevar acabo una tarea especifica, con mayor eficiencia y dependiendo del grado de inteligencia que disponga.

En los distintos laboratorios de investigación se está avanzando en los temas de visión por computadora, reconocimiento de patrones, respuesta ante estímulos externos, orientación y solución de problemas, entre otros, para que en un futuro los robots tengan formas humanas (autómatas).

Conclusiones:

Dado el marco teórico anterior se tiene un horizonte más amplio de las distintas funciones de los robots y hacia donde se dirije su evolución, tanto en el campo Laboral, como en el Técnico, Científico y Social.

A pesar del falso concepto de que todo este desarrollo se enfoca a un ámbito económico y comercial, con el propósito de agilizar tareas aún a costa de desplazar a trabajadores humanos, se debe resaltar que todos estos esfuerzos están dirigidos a la construcción de nuevas oportunidades que producirá el uso generalizado de los Robots en nuestra vida cotidiana, con el consecuente mejoramiento de nuestros niveles de vida.

Además, la tarea de remplazar totalmente a un trabajador humano abarca de lo difícil a lo imposible, ya que no sabemos como dotar a los robots de toda esa capacidad de percibir, razonar y actuar que tienen las personas. Así, más que un sustituto los Robots, con sus habilidades, constituyen un complemento del ser humano, y como tal, no se podría entender su existencia y funcionamiento sin el hombre o en un nivel superior a éste.

Referencias

[1] Lexipedia Tomo II. Encyclopaedia Britannica de México. 1989.

[2] Hunt, V."Smart Robots", Editorial Chapman & Hall. New York. 1986.

[3] Fu, K."Robótica: Control, detección, visión e Inteligencia". Editorial McGraw-Hill, Madrid, 1990.

[4] Sciavicco, L. "Modeling and Control of Robot Manipulators". Editorial McGraw-Hill, New York, 1996.

Oferta Educativa en Tecnología de Cómputo: Propuesta de una Maestría en el IPN

M. en C. Eduardo Rodríguez Escobar
Subdirector de Innovación y Desarrollo Tecnológico del CIDETEC-IPN

El Centro de Innovación y Desarrollo Tecnológico en Cómputo (CIDETEC) surge de la transformación del Centro de Investigación Tecnológica en Computación (CINTEC), a partir del acuerdo tomado por el Consejo General Consultivo del Instituto Politécnico Nacional en su reunión ordinaria de mayo de 1997, con los objetivos de consolidar la operación de la infraestructura de cómputo del Instituto, de desarrollar nuevas aplicaciones de la misma, de aprovechar las ventajas tecnológicas para incidir en el proceso productivo de bienes y servicios y de impartir cursos de capacitación, actualización, especialización y superación académica y profesional en el campo de la ingeniería de cómputo.

Con el fin de promover la superación profesional de los recursos humanos en el área de la Computación, se determinó incluir en el plan de trabajo 1998-2000 del Centro la oferta de un programa de Maestría, para ser impartida a partir del semestre de primavera de 1999. Esta determinación se fundó tanto en la necesidad de complementar y fortalecer las actividades que el Centro realiza en materia educativa, de investigación científica, de innovación tecnológica y de operación y mantenimiento de los sistemas y equipos de cómputo con que cuentan las distintas áreas, escuelas y centros del Instituto, como en la de integrar a la oferta educativa del Instituto Politécnico Nacional aquellas áreas que no se desarrollan en la medida de lo deseable en las escuelas y centros que ofrecen programas de posgrado en computación. Este objetivo no se pudo realizar en su momento debido a diversos cambios en la dirección del Centro, y por la falta de recursos humanos calificados en medida suficiente para implantar un programa de posgrado.

En este momento, sin embargo, el CIDETEC ya se encuentra en posibilidades de concretar estos objetivos. La propuesta que actualmente se esta desarrollando retoma el objetivo antes citado, revisado y actualizado con base en las necesidades tecnológicas de los sectores productivo y de servicios, como del mismo Instituto.

Objetivo

Formar recursos humanos en el área de la Ingeniería de Cómputo, así como especializar y actualizar a profesionales de la computación en esta área, que al egresar del programa contarán con los conocimientos y habilidades para desempeñar las siguientes actividades:

- Generar aplicaciones técnicas y científicas de la computación.
- Programar máquinas de alto rendimiento y aplicaciones de supercómputo.
- Implementar sistemas de medición y control computarizados para procesos industriales.
- Diseñar sistemas dedicados, tanto en lo que a hardware, como a software se refiere.
- Diseñar y programar sistemas de cómputo en el campo de la realidad virtual.
- Implementar sistemas de cómputo distribuido.

Justificación

En el contexto de los países miembros del Tratado de Libre Comercio, así como en la Comunidad Europea, se distinguen las curricula de Informática, Ciencias de la Computación, Ingeniería de Software e Ingeniería de Cómputo como áreas determinadas para objeto de estudio. En particular, la *Association for Computing Machinery* (ACM) y la *Computer Society* del *Institute for Electrical and Electronic Engineers* (IEEE-CS) definen a esta última como el análisis, diseño, construcción y la aplicación de computadoras y de sistemas digitales para la generación tanto de nuevas máquinas, como de aplicaciones dedicadas de la computación. La disciplina involucra tanto el hardware como el software, así como la interacción entre ambos. Un estudio comparativo de los programas de posgrado ofrecidos en nuestro país, presentado por este Centro a la Secretaría Académica, muestra claramente que la oferta en esta área es sumamente

pobre. El diseño de sistemas de cómputo dedicados y/o de propósito específico prácticamente no es cubierto en forma sistemática e integral. Así, aunque existen programas académicos relacionados con el diseño de sistemas digitales por una parte, y otros que se ocupan de la programación de aplicaciones técnicas y científicas, no existe un programa que integre estas áreas para aplicaciones desde medición, control de procesos, automatización industrial etc., hasta la generación de sistemas integrales dedicados en, por ejemplo, el diagnóstico médico.

Por otra parte, el creciente impacto de la educación no presencial en todas sus formas genera nuevas necesidades en materia de infraestructura computacional y de comunicaciones, así como en materia de programación en áreas tan computacionalmente intensivas como la realidad virtual. Estas nuevas aplicaciones requieren soluciones propias, no sólo por la importancia económica del mercado, sino por necesidades de control y óptimo aprovechamiento de la infraestructura sobre una base de disponibilidad cercana al 100%.

La presente propuesta pretende, así, llenar un vacío de la oferta educativa no sólo del Instituto, sino del país, en este campo de creciente importancia.

Vinculación con los Sectores Social, Productivo y de Servicios

El campo de acción de un egresado de esta maestría no se circunscribe a un conjunto pequeño y cerrado de aplicaciones, sino que incide en toda aquella actividad humana que haga uso de aplicaciones y sistemas computacionales. Como se mencionó en la justificación, prácticamente en cualquiera de los tres sectores (social, productivo y de servicios) existen necesidades y aplicaciones que exigen para su solución y realización de profesionales que cumplan con el perfil propuesto. La relación del programa con estos aspectos no se limita únicamente al campo de acción del alumno como egresado, sino que como parte de su formación y en apoyo a las funciones propias de vinculación y desarrollo tecnológico que el Centro tiene, los alumnos que se encuentren en las etapas finales del programa serán incorporados a aquellos proyectos que el Centro genere con cualquiera de los sectores mencionados, proporcionando la experiencia práctica y real necesaria para un desarrollo integral de sus habilidades y capacidades.

Perfil del egresado

Las características del egresado en términos de los conocimientos, habilidades académicas y aptitudes que alcanzará, son las siguientes:

- Poseer la habilidad de proyectar y diseñar aplicaciones específicas en materia de equipo de cómputo.
- Diseñar, integrar y operar sistemas de cómputo con base en subsistemas comercialmente disponibles, y componentes propietarias desarrolladas para el procesamiento de datos de propósito específico.
- Implementar sistemas de cómputo distribuido (redes).
- Capacidad para diseñar sistemas dedicados, tanto desde el punto de vista de hardware, como de software.
- Capacidad para desarrollar aplicaciones para equipos de cómputo de alto rendimiento, incluyendo la optimización de código, estimación y medición de rendimiento, evaluación del desempeño etc.).

Líneas de Investigación

El CIDETEC cuenta actualmente con dos líneas de investigación, ambas vinculadas en los diferentes aspectos y contenidos académicos y tecnológicos que conforman el programa de maestría propuesto. Los programas en cuestión, conjuntamente con la fundamentación de los mismos y sus proyectos integrantes son:

Cómputo de Alto Rendimiento para el Apoyo de la Investigación y Desarrollo Tecnológico en el IPN

Las computadoras paralelas de alto rendimiento computacional pueden clasificarse en dos grandes divisiones: máquinas de memoria compartida, en las que todos los procesadores tienen acceso a un solo espacio de direccionamiento común (compartido), y máquinas de memoria distribuida, en las cuales cada procesador accede únicamente a su memoria propia, y la interacción entre procesadores se obtiene mediante el intercambio de mensajes entre ellos. Esta categoría, que incluye los llamados "*clusters*", es efectiva para aquellos problemas computacionales que pueden ser resueltos con base en tareas independientes entre sí, que conllevan poco intercambio de información durante su ejecución, y que generan como resultado archivos grandes de información, requeridos por otros procesadores en etapas subsecuentes de ejecución (problemas de "granularidad gruesa"). Máquinas de memoria compartida, en cambio, aceptan modelos computacionales más flexibles, que requieren de mayores niveles de intercambio de información durante la ejecución de tareas individuales (problemas de "granularidad fina"). Una clase importante de computadoras paralelas de memoria compartida es la de acceso a memoria no uniforme, coherente a nivel de me-

moria caché (cc-NUMA: *cache-coherent, non-uniform memory access*). Esta clase de máquinas incluye modelos comerciales tan importantes como la Origin 2000 y 3000 de Silicon Graphics, Inc. La tecnología de memorias actualmente disponible excluye la posibilidad de que el acceso a memoria requiera la misma cantidad de tiempo para todos los procesadores a cualquier dirección (acceso no uniforme), por lo que la necesaria consistencia de contenidos de memoria para los diferentes niveles jerárquicos de la realización física (memoria caché de primer y segundo nivel, memoria principal) se asegura, en este tipo de máquinas, mediante un complejo protocolo basado en directorios distribuidos en los nodos computacionales que conforman la máquina.

La línea de investigación que se propone pretende resolver una serie de problemas que, hasta la fecha, han impedido que procesadores de la clase de máquinas personales, en particular, la familias Pentium e Itanium de Intel, constituyan los elementos computacionales de máquinas paralelas tipo cc_NUMA. Con ello, se genera la posibilidad de construir máquinas grandes (de cientos y hasta miles de procesadores) a partir de componentes y subsistemas económicos, comercialmente disponibles, y un número reducido de componentes propietarios que conforman los nodos de interconexión entre estos subsistemas. Tecnológicamente, dos desarrollos relativamente recientes apoyan esta posibilidad. Por una parte, la existencia de dispositivos programables de muy alta capacidad (más de un millón de compuertas por dispositivo, operando a más de 400 MHz) y, por otra, la disponibilidad de enlaces punto a punto ópticos operando a 20 GHz, permitiendo, en principio, esquemas de interconexión con un ancho de banda de más de 1 Gbyte/s.

Desde un punto de vista conceptual, máquinas del tipo cc_NUMA se enfrentan a problemas prácticos de realización física derivados del estado del arte tecnológico del momento. Estos problemas se pueden sintetizar en dos grandes rubros, relacionados entre sí: consistencia de memoria, por una parte, y latencia de acceso, por otra. El problema de consistencia se resuelve, como ya se mencionó, por medio de directorios distribuidos que permiten conocer la ubicación del dato más recientemente modificado correspondiente a una dirección de memoria dada en todos los nodos y niveles jerárquicos de memoria de la máquina. Existen una serie de protocolos de manejo de directorios correspondientes a diversos niveles de consistencia, definidos formalmente y que establecen modelos de programación para estas máquinas. El esquema de directorios permite, sin embargo, diferentes realizaciones físicas de distintos costos de memoria adicional requerida y con diferentes posibilidades de expansión (escalabilidad de la máquina). Recientemente, se ha propuesto un esquema basado en apuntadores que constituye un "standard" de la IEEE. Ninguno de los esquemas existentes, sin embargo, es enteramente satisfactorio y en todos los casos introduce un costo de tiempo importante para accesos a memoria que no pueden satisfacerse desde las memorias caché asociadas a los procesadores. Este costo de tiempo es parte de la latencia de acceso (tiempo que transcurre entre que un procesador emite una solicitud de acceso a memoria, y que ésta devuelva el primer byte de información solicitada). La latencia de acceso, por otra parte, depende también del número de nodos computacionales de la máquina, de la topología de la red de interconexión y de la lógica asociada a los circuitos de conmutación. Esto es, la latencia de acceso crece conforme la máquina cuenta con más nodos computacionales, lo que limita su escalabilidad y la vuelve menos efectiva para problemas computacionales de granularidad fina.

En términos de programación, existen lenguajes que ayudan en la paralelización de algoritmos escritos para ejecución secuencial, tales como el FORTRAN 95. Estos lenguajes suponen la existencia de determinada plataforma con características bien definidas, plataforma que es establecida por medio del sistema operativo y un programa de interfaz que, en conjunto, conforman el modelo de programación requerido. Actualmente, este *software* es, ya sea, propietario de una marca y para un tipo de equipo determinado, u obedece a alguna norma de uso generalizado, tal como puede ser el "*cluster*". También aquí, la relación entre el programa de aplicación y los dispositivos físicos establecida por varios niveles de programación introduce retardos en la ejecución y reduce el rendimiento de la máquina. Ello sugiere el empleo de un sistema operativo abierto, como el LINUX, y la generación de la interfaz requerida para el compilador a nivel de manejador de dispositivos, incorporados al sistema operativo. Con ello, se establece la posibilidad de optimizar, por diseño, el establecimiento de barreras, semáforos y la ejecución de algunos seudocomandos de sincronización, proporcionando determinados servicios en *hardware* y reduciendo el *software* de interfaz a un mínimo.

De lo anterior se desprende que el proyecto debe generar una solución al problema de cómputo de alto rendimiento, una necesidad de cuya satisfacción depende, en gran medida, la competitividad de la investigación y desarrollo tecnológico en muchas áreas del conocimiento que se cultivan en el IPN, involucra aspectos tanto formales, como tecnológicos de diseño de *hardware* y de *software*. Por otra parte, estos aspectos

están relacionados entre sí y los resultados obtenidos en una área afectan los objetivos y metas de las demás. Finalmente, la línea de trabajo que se propone constituiría un apoyo fundamental para un programa de maestría en ingeniería de cómputo. Este programa, a su vez, es una componente esencial en la relación entre oferta tecnológica, y aplicación de la misma en programas de investigación que requieren el apoyo computacional, pero no cuentan con los recursos humanos especializados en computación técnica y científica para aplicar estas herramientas.

Desarrollo de Aplicaciones Digitales y Soporte Didáctico al IPN

El programa que se presenta es un integración de parte de algunas de las labores de investigación, desarrollo tecnológico y apoyo académico que desde sus inicios como CINTEC y ahora como CIDETEC ha venido desarrollando este centro. Por un lado, el desarrollo tecnológico en base a los sistemas digitales es uno de los motivantes principales para la creación del Centro, y como tal, es una vocación de desarrollo firmemente respaldada por la evolución misma del Centro. Como respaldo de estos desarrollos puede recordarse simplemente el Programa de Autoequipamiento en materia de equipo de Cómputo, los desarrollos de microcomputadoras en base a microprocesadores Intel 8088, 80188, 80386 y Pentium, proyectos vinculados en el instituto de Investigaciones eléctricas, metro, tecnovisión de América, etc., todos ellos desarrollados con éxito en el CIDETEC. Actualmente se presenta como línea colateral un programa de Computo de altas prestaciones, que no es mas que la reafirmación de lo anterior.

Por otro lado, el soporte académico al Instituto se ha venido dado también desde sus inicios, desde la creación e impartición de una Maestría en Ingeniería de Cómputo, posteriormente transferida al CIC, la generación de cursos de capacitación en torno a arquitecturas de computadoras, el programa de capacitación de profesores de la DEMS en materia de cómputo, la presentación de sendas propuestas para el diseño y creación de una licenciatura en Ingeniería de Cómputo (ESCOM) y de un centro de Investigación en Cómputo (CIC), hasta la etapa actual de cursos y diplomados en el nivel superior y posgrado. Actualmente, las propuestas presentadas de apoyo académico se circunscriben al ámbito del apoyo por computadora, proponiendo esquemas interactivos alumno-sistema que puedan ser ejecutados en ambientes "stand alone", redes LAN Internet o Intranet.

Colaboración con otras instituciones educativas, productivas o de servicios

Las características del programa de maestría propuesto implican una interacción eficaz entre aquellas escuelas y centros del Instituto cuya labor académica gire alrededor de las Ciencias de la Computación. En particular, la presente propuesta plantea una colaboración inicial entre el CIDETEC, la UPIICSA, ESIME-CULHUACAN y el CIC, en los términos de establecer convenios específicos de colaboración e intercambio académico, permitiendo la participación de profesores visitantes en materias de especialización y electivas del programa propuesto, además de participar como asesores en proyectos y tesis de grado. Por otro lado, estos convenios permitirán a su vez la posibilidad de flexibilizar la currícula de la maestría, al homologar aquellas materias de especialidad y electivas cuyos contenidos sean equivalentes y que se impartan indistintamente en cualquiera de las escuelas y centros mencionados.