INSTITUTO TECNOLOGICO DE VERACRUZ

SIMULACION

ALUMNO: CARRO MENDEZ JOSE JAFET

TITULAR DE LA MATERIA: DR. JOSE ANTONIO GARRIDO NATAREN

H. VERACRUZ, VER. FEB. - JUN. 2015

Contenido

1. Teoría y Modelos de Simulación

1.1 Generalidades de modelos de simulación

La simulación es una técnica muy poderosa y ampliamente usada en las ciencias para analizar y estudiar sistemas complejos. En Investigaciones se formularon modelos que se resolvían en forma analítica. En casi todos estos modelos la meta era determinar soluciones óptimas. Sin embargo, debido a la complejidad, las relaciones estocásticas, etc., no todos los problemas del mundo real se pueden representar adecuadamente en forma de modelo. Cuando se intenta utilizar modelos analíticos para sistemas como éstos, en general necesitan de tantas hipótesis de simplificación que es probable que las soluciones no sean buenas, o bien, sean inadecuadas para su realización. En eso caso, con frecuencia la única opción de modelado y análisis de que dispone quien toma decisiones es la simulación. Simular, es reproducir artificialmente un fenómeno o las relaciones entrada−salida de un sistema.

¿Qué es el modelo de simulación?

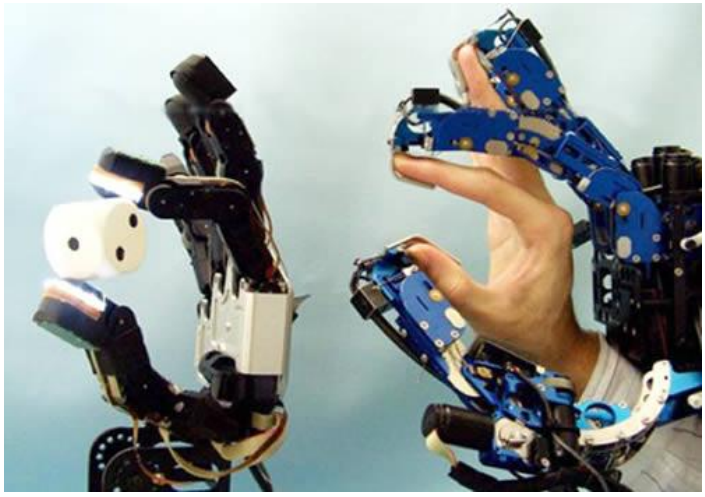
Es un instrumento que permite imitar el comportamiento de un sistema real mediante un artificio físico o matemático.

La simulación de sistemas implica la construcción de modelos. El objetivo es averiguar que pasaría en el sistema se acontecieran determinadas hipótesis. Desde muy antiguo la humanidad ha intentado adivinar el futuro, ha querido conocer que va a pasar cuando suceda un determinado hecho histórico. La simulación, ofrece sobre bases ciertas, esa predicción del futuro, condicionada a supuestos previos.

Para ello se construyen los modelos, normalmente una simplificación de la realidad.

Surgen de un análisis de todas las variables intervinientes en el sistema y de las relaciones que se descubren existen entre ellas.

A medida que avanza el estudio del sistema se incrementa el entendimiento que el analista tiene del modelo y ayuda a crear modelos más cercanos a la realidad. En el modelo se estudian los hechos salientes del sistema o proyecto. Se hace una abstracción de la realidad, representándose el sistema/proyecto, en un modelo.



Un modelo en sentido genérico es una representación simplificada de la realidad en la que aparecen

algunas de sus propiedades (Felicísimo, J. M. 1999), es decir, construcción de modelos donde se realiza el estudio con el fin de obtener conclusiones aplicables al sistema real.

Un modelo será tanto más representativo del fenómeno físico que simula, cuanto más capaz sea de reproducir su comportamiento, así como las leyes que lo rigen y sus interrelaciones con otros fenómenos.

Utilizar modelos de simulación facilita la compresión de las diferentes situaciones que se puedan presentar en la naturaleza, ya que permiten realizar una síntesis de los principales aspectos de los problemas; procurando satisfacer necesidades cambiantes de ese medio ambiente en el que esta insertado, resultando muy útiles para identificar los elementos más sensibles de un sistema y así, poder modificar su comportamiento y mejorar su eficiencia.

La simulación en sí, es el modelaje de un proceso o sistema de manera semejante que el modelo responda al sistema real tomando su lugar a través del tiempo. Para estudiar el comportamiento del modelo, tenemos que estudiar el comportamiento actual del sistema a estudiar.

Se requiere pues, que el modelo sea una fiel representación del sistema real. No obstante, el modelo no tiene por qué ser replica de aquel. Consiste en una descripción del sistema, junto con un conjunto de reglas que lo gobiernan. La descripción del sistema puede ser abstracta, física o simplemente verbal. Las reglas definen el aspecto dinámico del modelo. Se utilizan para estudiar el comportamiento del sistema real.

Como ejemplo de modelo físico se pueden citar los túneles de viento donde se ensayan los aviones, los simuladores de vuelo, los canales de experiencia donde se ensayan los barcos, etc.

1.2 Áreas de aplicación de la simulación

Actualmente la simulación presta un invalorable servicio en casi todas las áreas posibles, algunas de ellas son:

- **Procesos de manufacturas**: Ayuda a detectar cuellos de botellas, a distribuir personal, determinar la política de producción.
- **Plantas industriales**: Brinda información para establecer las condiciones óptimas de operación, y para la elaboración de procedimientos de operación y de emergencias.
- **Sistemas públicos**: Predice la demanda de energía durante las diferentes épocas del año, anticipa el comportamiento del clima, predice la forma de propagación de enfermedades.
- **Sistemas de transportes**: Detecta zonas de posible congestionamiento, zonas con mayor riesgo de accidentes, predice la demanda para cada hora del día.
- **Construcción**: Predice el efecto de los vientos y temblores sobre la estabilidad de los edificios, provee información sobre las condiciones de iluminación y condiciones ambientales en el interior de los mismos, detecta las partes de las estructuras que deben ser reforzadas.
- **Diseño**: Permite la selección adecuada de materiales y formas. Posibilita estudiar la sensibilidad del diseño con respecto a parámetros no controlables.
- **Educación**: Es una excelente herramienta para ayudar a comprender un sistema real debido a que puede expandir, comprimir o detener el tiempo, y además es capaz de brindar información sobre variables que no pueden ser medidas en el sistema real.
- **Capacitación**: Dado que el riesgo y los costos son casi nulos, una persona puede utilizar el simulador para aprender por sí misma utilizando el método más natural para aprender: el de prueba y error.

Las áreas de aplicación de la simulación son muy amplias, numerosas y diversas, basta mencionar sólo algunas de ellas:

- Análisis del impacto ambiental causado por diversas fuentes Análisis y diseño de sistemas de manufactura Análisis y diseño de sistemas de comunicaciones.
- Evaluación del diseño de organismos prestadores de servicios públicos (por ejemplo: hospitales, oficinas de correos, telégrafos, casas de cambio, etc.).
- Análisis de sistemas de transporte terrestre, marítimo o por aire.
- Análisis de grandes equipos de cómputo.
- Análisis de un departamento dentro de una fábrica. 0
- Adiestramiento de operadores (centrales carboeléctricas, termoeléctricas, nucleoeléctricas, aviones, etc.).
- Análisis de sistemas de acondicionamiento de aire.
- Planeación para la producción de bienes.
- Análisis financiero de sistemas económicos.
- Evaluación de sistemas tácticos o de defensa militar.

La simulación se utiliza en la etapa de diseño para auxiliar en el logro o mejoramiento de un proceso o diseño o bien a un sistema ya existente para explorar algunas modificaciones. Se recomienda la aplicación de la simulación a sistemas ya existentes cuando existe algún problema de operación o bien cuando se requiere llevar a cabo una mejora en el comportamiento. El efecto que sobre el sistema ocurre cuando se cambia alguno de sus componentes se puede examinar antes de que ocurra el cambio físico en la planta para asegurar que el problema de operación se soluciona o bien para determinar el medio más económico para lograr la mejora deseada. Todos los modelos de simulación se llaman modelos de entrada-salida. Es decir, producen la salida del sistema si se les da la entrada a sus subsistemas interactuantes. Por tanto los modelos de simulación se "corren" en vez de "resolverse", a fin de obtener la información o los resultados deseados. Son incapaces de generar una solución por si mismos en el sentido de los modelos analíticos; solos pueden servir como herramienta para el análisis del comportamiento de un sistema en condiciones especificadas por el experimentador. Por tanto la simulación es una teoría, sino una metodología de resolución de problemas. Además la simulación es solo uno de varios planteamientos valiosos para resolver problemas que están disponibles para el análisis de sistemas.

Pero ¿Cuándo es útil utilizar la simulación?

Cuando existan una o más de las siguientes condiciones:

1.- No existe una completa formulación matemática del problema o los métodos analíticos para resolver el modelo matemático no se han desarrollado aún. Muchos modelos de líneas de espera corresponden a esta categoría.

2.- Los métodos analíticos están disponibles, pero los procedimientos matemáticos son tan complejos y difíciles, que la simulación proporciona un método más simple de solución.

3.- Las soluciones analíticas existen y son posibles, pero están mas allá de la habilidad matemática del personal disponible El costo del diseño, la prueba y la corrida de una simulación debe entonces evaluarse contra el costo de obtener ayuda externa.

4.- Se desea observar el trayecto histórico simulado del proceso sobre un período, además de estimar ciertos parámetros.

5.- La simulación puede ser la única posibilidad, debido a la dificultad para realizar experimentos y observar fenómenos en su entorno real, por ejemplo, estudios de vehículos espaciales en sus vuelos interplanetarios.

6.- Se requiere la aceleración del tiempo para sistemas o procesos que requieren de largo tiempo para realizarse. La simulación proporciona un control sobre el tiempo, debido a que un fenómeno se puede acelerar o retardar según se desee.

1.3 Concepto de Sistema

Sistema
Pueden darse varias definiciones de sistema:
"Conjunto de **elementos** cuya **interacción** interesa estudiar"

"Conjunto de **elementos** que **interactúan** entre sí, con un fin común, que se aísla del universo para su estudio."

"Conjunto de **partes** organizado funcionalmente de manera tal de constituir una unidad"

Interconectada

"Conjunto de **elementos que interactúan entre ellos" Pierre Delattre 1971."**

Subsistema

Es un conjunto que se aísla dentro del sistema. El sistema puede verse como un subsistema del universo.

Cada subsistema puede ser tratado dentro del sistema o estudiado en forma aislada.
El **comportamiento** del sistema total depende de:

1. El comportamiento de cada **subsistema.**
2. Las **relaciones** entre los subsistemas.
3. Las relaciones con el mundo exterior, o sea con el **medioambiente** que lo circunda.

1.4 Concepto de Modelo

Una definición bastante generalizada de modelo, es “una representación simplificada de la realidad en la que aparecen algunas de sus propiedades.

De la definición se deduce que la versión de la realidad que se realiza a través de un modelo pretende reproducir solamente algunas propiedades del objeto o sistema original que queda representado por otro objeto o sistema de menor complejidad.

“un modelo es un objeto, concepto o conjunto de relaciones que se utiliza para representar y estudiar de forma simple y comprensible una porción de la realidad empírica”

Para que los modelos puedan representarse, es necesario que se construyan estableciendo una relación con la realidad que debe ser simétrica, es decir, la relación de correspondencia entre el objeto real y el modelo debe ser lo menos parcialmente reversible y debe permitir la traducción de algunas propiedades de modelo a la realidad.

CLASIFICACIÓN DE LOS MODELOS

Existen múltiples tipos de modelos para representar la realidad. Algunos de ellos son:

- **Dinámicos:** Utilizados para representar sistemas cuyo estado varía con el tiempo.
- **Estáticos:** Utilizados para representar sistemas cuyo estado es invariable a través del tiempo.
- **Matemáticos:** Representan la realidad en forma abstracta de muy diversas maneras.
- **Físicos:** Son aquellos en que la realidad es representada por algo tangible, construido en escala o que por lo menos se comporta en forma análoga a esa realidad (maquetas, prototipos, modelos analógicos, etc.).
- **Analíticos:** La realidad se representa por fórmulas matemáticas. Estudiar el sistema consiste en operar con esas fórmulas matemáticas (resolución de ecuaciones).
- **Numéricos:** Se tiene el comportamiento numérico de las variables intervinientes. No se obtiene ninguna solución analítica.
- **Continuos:** Representan sistemas cuyos cambios de estado son graduales. Las variables intervinientes son continuas.
- **Discretos:** Representan sistemas cuyos cambios de estado son de a saltos. Las variables varían en forma discontinua.
- **Determinísticos:** Son modelos cuya solución para determinadas condiciones es única y siempre la misma.

- **Estocásticos:** Representan sistemas donde los hechos suceden al azar, lo cual no es repetitivo. No se puede asegurar cuáles acciones ocurren en un determinado instante. Se conoce la probabilidad de ocurrencia y su distribución probabilística.

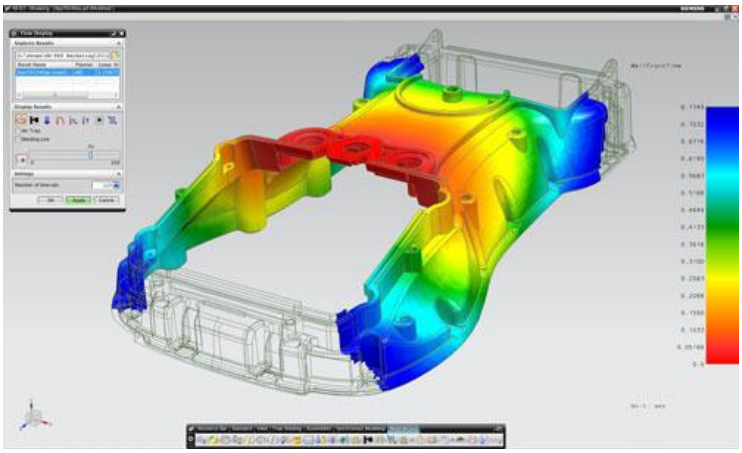
MODELOS DE SIMULACION ESTATICA VS. DINAMICA

Un modelo de simulación estática, se entiende como la representación de un sistema para un instante (en el tiempo) en particular o bien para representar un sistema en el que el tiempo no es importante, por ejemplo la simulación Montecarlo; en cambio un modelo de simulación dinámica representa a un sistema en el que el tiempo es una variable de interés, como por ejemplo en el sistema de transporte de materiales dentro de una fábrica, una torre de enfriamiento de una central termoeléctrica, etc.

MODELOS DE SIMULACION DETERMINISTA VS. ESTOCASTICA

Si un modelo de simulación no considera ninguna variable importante, comportándose de acuerdo con una ley probabilística, se le llama un modelo de simulación determinista. En estos modelos la salida queda determinada una vez que se especifican los datos y relaciones de entrada al modelo, tomando una cierta cantidad de tiempo de cómputo para su evaluación. Sin embargo, muchos sistemas se modelan tomando en cuenta algún componente aleatorio de entrada, lo que da la característica de modelo estocástico de simulación.

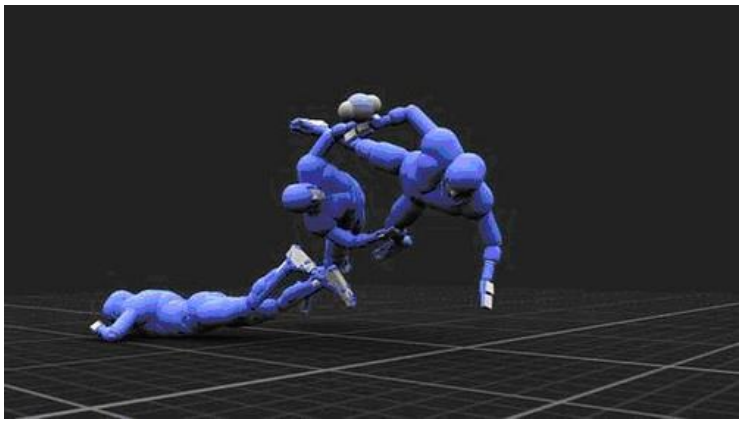
Un ejemplo sería un sistema de inventarios de una fábrica, o bien el sistema de líneas de espera de una fábrica, etc. estos modelos producen una salida que es en sí misma de carácter aleatorio y ésta debe ser tratada únicamente para estimar las características reales del modelo, esta es una de las principales desventajas de este tipo de simulación.



MODELOS DE SIMULACION CONTINUOS VS. DISCRETOS

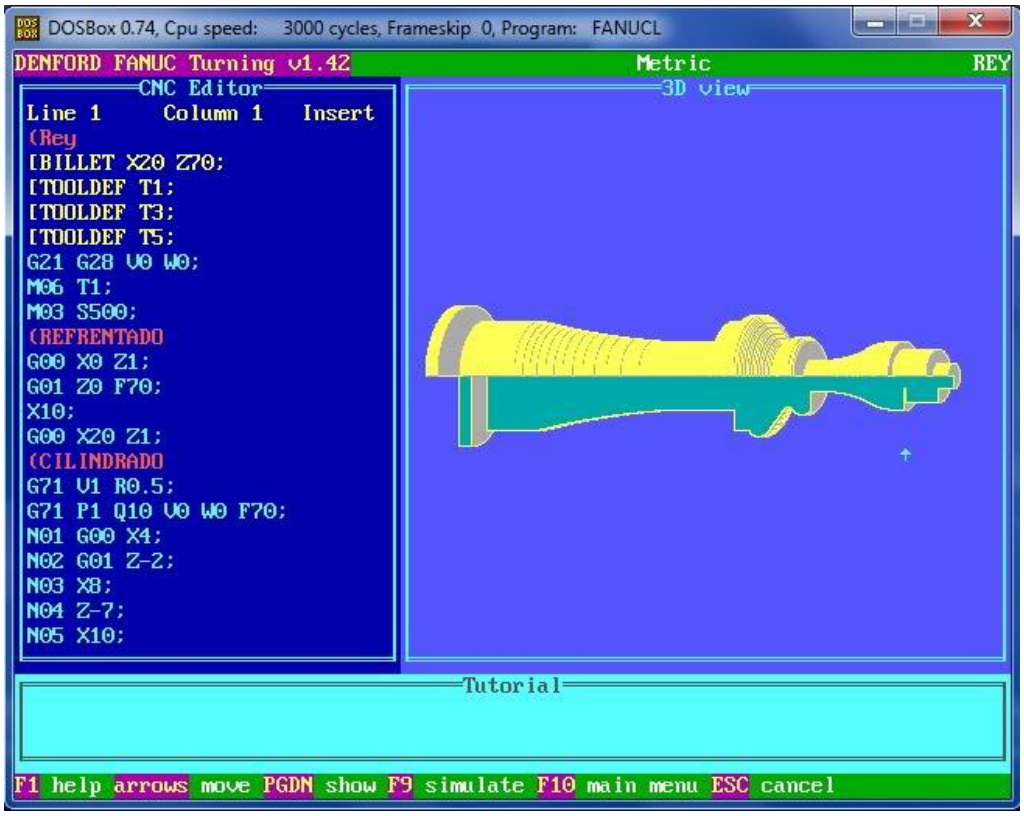
Los modelos de simulación discretos y continuos, se definen de manera análoga a los sistemas discretos y continuos respectivamente. Pero debe entenderse que un modelo discreto de simulación no siempre se usa para modelar un sistema discreto. La decisión de utilizar un modelo discreto o continuo para simular un sistema en particular, depende de los objetivos específicos de estudio. Por ejemplo: un modelo de flujo de tráfico en una supercarretera, puede ser discreto si las características y movimientos de los vehículos en forma individual es importante. en cambio si los vehículos pueden considerarse como un agregado en el flujo de tráfico entonces se puede usar un modelo basado en ecuaciones diferenciales presentes en un modelo continuo.

Algunos ejemplo de simulación:



Simulación de movimientos dinámicos de jugadores de futbol americano.

Simulador de aviones de guerra.



Simulador de escáner digital de piezas en 3D.

1.5 Proceso de Simulación

1. Definición del sistema con el máximo de detalle

- Se debe evitar comenzar a trabajar en la construcción del modelo con un sistema superficial, mal concebido. ¡Se perderán horas hombre y de computadora en tares inútiles!
- Es un principio comprobado de organización que la incidencia de un error en un proyecto aumenta dramáticamente con el instante en que se lo descubre. Es decir, cuánto más se demora en detectarlo mucho más complicada es su corrección.
- Se debe discutir en detalle el sistema; analista y usuario reunidos durante largas horas evitarán que el sistema tenga que ser redefinido después.
- En esta etapa se definen los límites del sistema y los objetivos del estudio, chequeando que estos no cambien durante el desarrollo del mismo.
- Deben tenerse en cuenta las condiciones iníciales del sistema y sus condiciones de régimen. Interesa estudiarlo ya en régimen y no inicialmente cuando los recursos están desocupados y favorecen el movimiento de los elementos por el sistema. El modelo debe considerar qué resultados

estadísticos interesan obtenerse para evaluar correctamente al sistema en estudio.

Ejemplos: tiempos en cola, longitudes de las colas que se forman en los distintos sectores, tiempo que está cada cliente en el sistema, promedios, desviaciones estándar, etc.

2. Elección del método para realizar el estudio

- Búsqueda de la herramienta analítica de resolución.
- Adopción de la misma en caso de encontrarla.
- Utilización de la simulación como última alternativa.

3. Variables a incluir en el modelo

¿Qué variables, parámetros se incluyen? ¿Cuáles se desprecian por su irrelevancia?
La elección no es sencilla.
Conviene hacer un ranking de las variables y restricciones del sistema en orden de importancia. Este ranking debe ser discutido con el usuario y con los distintos especialistas a fin de proceder a su verificación y eventual corrección.
Se debe recordar que quitar una variable superflua de un sistema es algo bastante sencillo, mientras que incluir una que se había despreciado es de ordinario mucho más complicado. Tomar debida cuenta de los casos especiales muchas veces estos obligan a tener en cuenta variables despreciables para el resto de los casos.
Esta selección de variables a considerar depende de la mecánica con que se maneja el sistema, de la experiencia que se tenga de él e incluso de la intuición del grupo humano que interviene en el estudio.

Se debe evitar una sobre simplificación que inválida al modelo en cuanto se lo quiere ensayar con casos especiales, o una sobre especificación que hace largo y difícil el trabajo de construir el modelo.
Todas las variables que intervienen en un modelo son medibles. No siempre es posible lo mismo con las que intervienen en un sistema real. Muchas veces se debe hacer una estimación de las mismas con el fin de incorporarlas en el modelo.
Existen variables endógenas (internas y controladas por el sistema) y exógenas (externas al sistema y fuera de su control).
Existen variables cualitativas, como la preferencia personal, y cuantitativas como la frecuencia con que arriban los clientes a un banco. Todas deben ser estimadas en términos cuantitativos.

4. Recolección y análisis de los datos del sistema

Definidas las variables intervinientes en el sistema es habitual que existan muchas variables estocásticas.
Para esas variables se debe disponer de:

- La densidad de probabilidad o

- La función de distribución acumulativa en forma matemática o

- Una tabla de valores del comportamiento de la variable.

Se utiliza para ello todas las herramientas estadísticas clásicas, tales como, análisis de regresión, de serie de tiempos y de varianzas.
Se debe hacer un relevamiento del tiempo que se insume en las distintas tareas tratando de no obtener datos distorsionados producto de la medición (la persona trabaja más rápido o más lento debido a que lo están midiendo y le parece más conveniente mostrarse en forma distorsionada).
Si se tiene el valor medio de una medición y no se conoce su distribución, es preferible adoptar una distribución exponencial que una uniforme, pues en la primera, pueden darse situaciones críticas que no se dan en la segunda.
El tiempo empleado validando los datos de entrada está totalmente justificado y es absolutamente necesario para construir un modelo válido sobre el cual se puedan sacar conclusiones aplicables al sistema real.

5. Definición de la estructura del modelo

Se definen:
- Las entidades permanentes y sus atributos, es decir, los recursos con que se cuenta en el sistema y cuantitativamente cómo es su comportamiento.

- Las entidades transitorias que circulan por el modelo tienen definida probabilísticamente su ruta por el sistema y los tiempos de utilización de los recursos.

- Los eventos que provocan los cambios de estado, modificando los atributos de las entidades.

Se debe diseñar el modelo de manera que los cambios en su estructura estén en cierto modo previstos.

6. Programación del modelo

Obtención del programa de computadora que representa el modelo. Se debe elegir el lenguaje con que se construirá el modelo; Una vez elegido, se lo utiliza para construir el modelo, que debe representar fielmente todo lo que ha sido relevado del sistema.

7. Validación del modelo

Aunque imposible de demostrar rigurosamente se trata de verificar al modelo con una serie de situaciones conocidas como para tener un alto grado de confiabilidad.

8. Análisis y crítica de los resultados

Paso previo a la entrega de resultados al usuario se debe:

- Verificar que los resultados obtenidos sean realmente suficientes para tomar una correcta decisión.

- Hacer una buena compactación en la presentación de los mismos procurando quesean perfectamente comprensibles para el usuario.

- Recordar que un exceso de información ocasiona casi los mismos inconvenientes que la falta de información, ya que el usuario en ambos casos no puede acceder a los resultados que necesita como apoyo a la toma de decisiones (en un caso porque no sabe cómo accederlos, en el otro porque no los tiene).

- Estudiar la factibilidad, y, en caso afirmativo, proponer una alternativa que signifique un cambio estructural del sistema y por ende del modelo, la que se considera digna de tener en cuenta antes de tomar una decisión definitiva.

2 Las Etapas de la Simulación Numérica

2.1 La Formulación del problema

En este paso debe quedar perfectamente establecido el objeto de la simulación.

- los resultados que se esperan del simulador
- el plan de experimentación
- el tiempo disponible
- las variables de interés
- el tipo de perturbaciones a estudiar
- el tratamiento estadístico de los resultados
- la complejidad de la interfaz del simulador

Se debe establecer si el simulador será operado por el usuario o si el usuario sólo recibirá los resultados.

2.2 La Definición del Sistema y la Formulación del Modelo

1) **La definición del sistema** consiste en identificar los objetivos del proyecto, especificar los índices de medición de la efectividad del sistema, establecer los objetivos específicos del modelamiento y definir el sistema que se va a modelar. Hacer un análisis preliminar del mismo, con el fin de determinar la interacción del sistema con otros sistemas, las restricciones del sistema, las variables que interactúan dentro del sistema y sus interrelaciones, las medidas de efectividad que se van a utilizar para definir y estudiar el sistema y los resultados que se esperan obtener del estudio.

El sistema a simular debe estar perfectamente definido. El cliente y el desarrollador deben acordar dónde estará la frontera del sistema a estudiar y las interacciones con el medioambiente que serán consideradas

2) **La Formulación del Modelo** Comienza con el desarrollo de un modelo simple que captura los aspectos relevantes del sistema real. Los aspectos relevantes del sistema real dependen de la formulación del problema, en la formulación del modelo es necesario definir todas las variables que forman parte de él, sus relaciones lógicas y los diagramas de flujo que describan en forma completa el modelo. 2 El desarrollo del modelo debe contemplar los aspectos relevantes del sistema real que dependen de la formulación del problema. Es necesario definir todas las variables que forman parte de él, sus relaciones lógicas y los diagramas de flujo que describan en forma completa al modelo.

2.3 Colección de datos y la implementación del modelo

3) Es importante que se definan con claridad y exactitud los datos que el modelo va a requerir para producir los resultados deseados. La naturaleza y cantidad de datos necesarios están determinadas por la formulación del problema y del modelo. Los datos pueden ser provistos por:

- Registros históricos
- Experimentos de laboratorios
- Mediciones realizadas en el sistema real.

Los mismos deberán ser procesados adecuadamente para darles el formato exigido por el modelo.

4) El modelo es implementado en la computadora utilizando algún lenguaje de computación. Existen lenguajes específicos de simulación que facilitan esta tarea; también, existen programas que ya cuentan con modelos implementados para casos especiales.

2.4 La Verificación, la Validación y el Diseño de Experimentos

5) El proceso de verificación consiste en comprobar que el modelo simulado cumple con los requisitos de diseño para los que se elaboró. Se trata de evaluar que el modelo se comporta de acuerdo a su diseño del modelo.

En esta etapa se comprueba que no se hayan cometidos errores durante la implementación del modelo. Para ello, se utilizan las herramientas de *debugging* provistas por el entorno de programación.

6) En la siguiente etapa, la validación, se valoran las diferencias entre el funcionamiento del simulador y el sistema real que se está tratando de simular. Las formas más comunes de validar un modelo son:

1. La opinión de expertos sobre los resultados de la simulación.

2. La exactitud con que se predicen datos históricos.

3. La exactitud en la predicción del futuro.

4. La comprobación de falla del modelo de simulación al utilizar datos que hacen fallar al sistema real.

5. La aceptación y confianza en el modelo de la persona que hará uso de los resultados que arroje el experimento de simulación.

Se comprueba la exactitud del modelo desarrollado. Esto se lleva a cabo comparando las predicciones del modelo con:

- Mediciones realizadas en el sistema real
- Datos históricos o datos de sistemas similares.

Como resultado de esta etapa puede surgir la necesidad de modificar el modelo o recolectar datos adicionales.

7) **El diseño de experimento**Consiste en generar bitácoras de prueba a las que deberá de someterse el modelo. Las pruebas son necesarias pues solo ellas permiten validar que el modelo funcionará bajo las condiciones a las que fue probado. Los datos de prueba son determinados por los requerimientos del sistema a simular haciendo de acuerdo a un análisis de sensibilidad de los índices requeridos.

En esta etapa se decide las características de los experimentos a realizar:

1. el tiempo de arranque
2. el tiempo de simulación
3. el número de simulaciones.

Debe quedar claro cuando se formula el problema si lo que el cliente desea es un estudio de simulación o de optimización.

2.5 La implementación, la experimentación, la interpretación y la documentación

8) Conviene acompañar al cliente en **la etapa de implementación** para evitar el mal manejo del simulador o el mal empleo de los resultados del mismo.

9) **La experimentación** con el modelo se realiza después que este haya sido validado. La experimentación consiste en comprobar los datos generados como deseados y en realizar un análisis de sensibilidad de los índices requeridos. En esta etapa se realizan las simulaciones de acuerdo el diseño previo. Los resultados obtenidos son debidamente recolectados y procesados.

10) **La interpretación** no muestra los resultados que arroja la simulación y con base a esto se toma una decisión. Se analiza la sensibilidad del modelo con respecto a los parámetros que tienen asociados la mayor incertidumbre. Si es necesario, se deberán recolectar datos adicionales para refinar la estimación de los parámetros críticos.

11) **En la documentación**, incluye la elaboración de la documentación técnica y manuales de uso. La documentación técnica debe contar con una descripción detallada del modelo y de los datos. También, se debe incluir la evolución histórica de las distintas etapas del desarrollo. Esta documentación será de utilidad para el posterior perfeccionamiento del simulador. Dos tipos de documentación son requeridos para hacer un mejor uso del modelo de simulación. La primera se refiere a la documentación del tipo técnico y la segunda se refiere al manual del usuario, con el cual se facilita la interacción y el uso del modelo desarrollado.

4 Herramientas de Programación de Modelos de Simulación Numérica

4.1 Lenguajes de programación para la simulación

Un lenguaje de simulación es un software con características de lenguaje general pero con adaptaciones especiales para facilitar la realización de aplicaciones específicas, estas adaptaciones incluyen metodologías apoyadas en librerías y símbolos propios de la simulación y el entorno a modelar.

En general dan al usuario un conjunto de conceptos de modelado para describir el sistema y un sistema de programación para convertir la descripción en un programa que ejecuta la simulación, además de facilidades para la detección automática de errores potenciales que ya han sido identificados.

En un principio, los programas de simulación se elaboraban utilizando algún lenguaje de propósito general, como ASSEMBLER, FORTRAN, ALGOL o PL/I. A partir de la década de 1960 hacen su aparición los lenguajes específicos para simulación como GPSS, GASP, SIMSCRIPT, SLAM. En la última década del siglo pasado la aparición de las interfaces gráficas revolucionaron el campo de las aplicaciones en esta área, y ocasionaron el nacimiento de los simuladores.

En lo práctico, es importante utilizar la aplicación que mejor se adecúe al tipo de sistema a simular, ya que de la selección del lenguaje o simulador dependerá el tiempo de desarrollo del modelo de simulación. Las opciones van desde las hojas de cálculo, lenguajes de tipo general (como **Visual Basic**, **C++** o **Fortan**), lenguajes específicos de simulación (como **GPSS**, **SLAM**, **SIMAN**, **SIMSCRIPT**, **GAS** y **SSED**), hasta simuladores específicamente desarrollados para diferentes objetivos (como **SIMPROCESS**, **ProModel**, **Witness**, **Taylor II** y **Cristal Ball**).

Para procesar “Transacciones” (objetos) en espera de un ordenamiento, un lenguaje de simulación debe proporcionar un medio automático de almacenamiento y recuperación de estas entidades. Atendiendo a la orientación del modelamiento de una simulación existen tres formas:

1.- ***Programación de eventos***.- en esta se modela identificando las características del evento y programando las rutinas a realizar para cada uno de los eventos definidos, detallando los cambios que ocurren en el tiempo en cada evento. SLAM usa este tipo de programación.

2.- ***Por procesos***.- Consta de una secuencia de tiempos interrelacionados donde se describe la experiencia de una entidad a través del sistema. Ejemplo: SIMAN.
3.- ***Por Examinación de actividades***.- En este método, el modelador define las condiciones necesarias al inicio y fin de cada actividad del sistema, manejando el avance de tiempo en intervalos regulares y evaluando en cada intervalo las condiciones del sistema para determinar si alguna actividad debe estar iniciando o terminando.

Ventajas
Produce un código más legible y más fácil de modificar ya que proporcionan automáticamente las características necesarias para la programación de un modelo de simulación, lo que redunda en una reducción del esfuerzo requerido para programar, al proporcionar un marco de trabajo más natural con bloques básicos de construcción mucho más adecuados para el modelado de sistemas que un lenguaje de tipo general. Algunos de estos lenguajes contienen librerías y aplicaciones que facilitan las tareas de comunicación, manipulación online de modelos, etc. *U.4. Herramientas de programación de modelos de simulación numéric*a. *4.1 Lenguajes de programación para la simulación.*

Desventajas

Estos lenguajes requieren de cierto conocimiento acerca de programación y algo de familiarización con la estructura y desarrollo, su flexibilidad puede verse parcialmente limitada por la validez del modelo y deben ser seleccionados de acuerdo a la tarea que se va a realizar y las especificaciones que esta tenga.

Ejemplos

GPSS.- Fue creado por IBM en 1961 y es un lenguaje altamente estructurado, un lenguaje de simulación de propósito especial que usa en el enfoque basado en procesos y se orienta hacia los sistemas de colas. Un diagrama de bloques provee una forma conveniente para describir el sistema que se está simulando. Las entidades, llamadas transacciones, pueden ser vistas como que fluyen a través de un diagrama de bloques. Por lo anterior, GPSS puede ser usado para modelar una situación donde las transacciones (entidades, clientes, unidades de tráfico) están fluyendo a través del sistema.

SIMAN.- Por sus siglas en ingles SIMAN significa Análisis, modelación y simulación (Simulation Modeling and Análisis). Las capacidades del lenguaje incluyen orientación basada en procesos, orientación basada en eventos, y simulación continua, o una mezcla de estas.

Características destacadas

- Habilidad de describir el medio ambiente de los centros de trabajo (estaciones) y la habilidad de definir una secuencia de entidades en movimiento a través del sistema.
- Permiten modelar sistemas de manejo de materiales.
- Un controlador de corridas interactivo que permite puntos de cambio, relojes, y otros procedimientos de ejecución.
- La accesibilidad del modelo a todo tipo de computadoras.

GASP IV.- Es una colección de subrutinas de Fortran diseñadas para facilitar la simulación de secuencia de eventos. Cerca de 30 subrutinas y funciones que proveen numerosas facilidades, entre ellas:

- Rutinas de avance del tiempo.
- Adición y remoción de entidades.
- Colección de estadísticas.
- Generadores de variables aleatorias.
- Reporte estándar.

Los lenguajes de simulación facilitan enormemente el desarrollo y ejecución de simulaciones de sistemas complejos del mundo real. Los lenguajes de simulación son similares a los lenguajes de programación de alto nivel pero están especialmente preparados para determinadas aplicaciones de la simulación. Así suelen venir acompañados de una metodología de programación apoyada por un sistema de símbolos propios para la descripción del modelo por ejemplo mediante diagramas de flujo u otras herramientas que simplifican notablemente la modelización y facilitan la posterior depuración del modelo.

Características de los lenguajes de simulación:

- Los lenguajes de simulación proporcionan automáticamente las características necesarias para la programación de un modelo de simulación, lo que redunda en una reducción significativa del esfuerzo requerido para programar el modelo.
- Proporcionan un marco de trabajo natural para el uso de modelos de simulación. Los bloques básicos de construcción del lenguaje son mucho más afines a los propósitos de la simulación que los de un lenguaje de tipo general.
- Los modelos de simulación son mucho más fácilmente modificables.
- Proporcionan muchos de ellos una asignación dinámica de memoria durante la ejecución.
- Facilitan una mejor detección de los errores.
- Los paquetes de software especialmente diseñados para simulación contienen aplicaciones diversas que facilitan al simulador las tareas de comunicaciones, la depuración de errores sintácticos y de otro tipo de errores, la generación de escenarios, la manipulación “on-line” de los modelos, etc.
- Son muy conocidos y en uso actualmente
- Aprendizaje lleva cierto tiempo
- Simuladores de alto nivel
- Muy fáciles de usar por su interface gráfica
- Restringidos a las áreas de manufactura y comunicaciones
- Flexibilidad restringida puede afectar la validez del modelo

Ventajas

Entre las *ventajas* del empleo de estos lenguajes están su velocidad, portabilidad y flexibilidad, ya que permiten crear programas compatibles con diversos sistemas operativos, además, presentan la posibilidad de realizar proyectos a la medida de la necesidad, lo que evita “cargar” con herramientas o prestaciones inútiles para el propósito que requerimos, haciendo las simulaciones menos pesadas y más manejables, así como también, las simulaciones creadas en estos lenguajes pueden ser modificadas y ampliadas hasta donde se requiera prácticamente sin limitaciones, pues cuentan con librerías para la realización de tareas repetitivas o de uso común, todo esto con el fin de facilitar la programación.

Desventajas
Por otra parte, el tiempo de desarrollo de las aplicaciones es más largo que en herramientas específicas y la programación requiere de conocimientos más específicos sobre el funcionamiento y estructura de programación.
Ejemplos

En este grupo se encuentran lenguajes como: C, Fortran, Basic, Cobol, Algol, Pascal, etc.

4.2 Programas y Algoritmos

Entre estos lenguajes específicos podemos nombrar los siguientes:

MIDAS, DYSAC, DSL , GASP, MIMIC, DYNAMO, GPSS, SIMULA, CSSL(Continuos System Simulation Language) , CSMP, ACSL (Advanced Continuos Simulation Language), DARE-P and DARE-Interactive, C-Simscript, SLAM, SIMAN, SIMNON, SIMSCRIPT-II-5, ADA, GASP IV, SDL.

Muchos de estos lenguajes dependen fuertemente de los lenguajes de propósito general como es el caso de SLAM o SIMAN que dependen de Fortran para las subrutinas.
Por otro lado, el GPSS es un caso especial de un lenguaje de simulación de propósito especial, altamente estructurado que está orientado a la transacción, un caso especial de una orientación basada en procesos más general. El GPSS fue diseñado para la simulación simple de sistemas de colas tales como trabajos de taller. A diferencia de los otros lenguajes de simulación, GPSS tiene varias implementaciones incluyendo GPSS/H y GPSS/PC, ambos de los cuales serán discutidos más adelante. El SIMAN V, SIMSCRIPT II.5, y el SLAM son lenguajes de simulación de alto nivel que tienen constructor especialmente diseñados para facilitar la construcción de modelos. Estos lenguajes proveen al analista de simulación con una opción orientación basada en procesos o basada en eventos, o un modelo usando una mezcla de las dos orientaciones. A diferencia del FORTRAN, estos tres lenguajes proveen la administración de la lista de eventos futuros, generador interno de variables aleatorias, y rutinas internas para la obtención de estadísticas (estas características para las implementaciones del GPSS mencionadas previamente.) Se pueden lograr cálculo complejos en ambas implementaciones del GPSS y estos tres lenguajes. El SIMAN, SIMSCRIPT II.5, y el SLAMSYSTEM proveen la capacidad de realizar simulación continua (esto es, para modelar sistemas que tengan continuamente cambios en sus variables de estado) pero este tema no está dentro del alance de este libro.

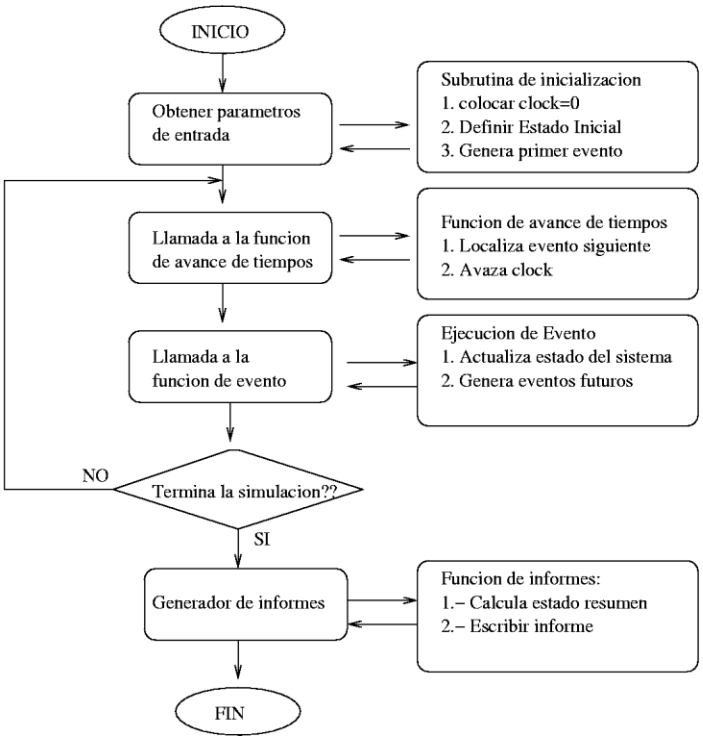
El SIMAN está escrito en C, aunque las primeras versiones del lenguaje fue escrito en FORTRAN.
El SIMAN V puede ser acezado directamente, o a través del medio ambiente del ARENA.
El SLAMSYSTEM contiene al lenguaje de simulación SLAM II. El SLAM II está basado en

el FORTRAN y contiene al lenguaje GASP como un subconjunto. El GASP es un conjunto de subrutinas en FORTRAN para facilitar las simulaciones orientadas al objeto escritas en FORTRAN. El SIMSCRIPT II.5 por otro lado, contiene un subconjunto de un completo lenguaje científico de simulación comparable con el FORTRAN, C o C++. El MODSIM III es un descendiente del lenguaje que la compañía de productos CACI originalmente diseñado por la armada de los Estados Unidos. Hereda mucha de su sintaxis del MODULA-2 y del ADA, ciertas características del ADA y sus conceptos de simulación del SIMSCRIPT y el SIMULA. Algunas de las características de la simulación orientada al objeto fueron originalmente vistas en el SIMULA y el SMALLTALK.
Un claro ejemplo de uno de los leguajes de programación citados y su funcionamiento.

USO DE UN LENGUAJE DE PROGRAMACION
EN SIMULACION DE EVENTOS DISCRETOS.

ELEMENTOS DEL PROGRAMA
- Funcion de inicializacion.
- Funcion de avance de tiempos
- Funcion de planificacion
- Funciones de eventos
- Generadores de variables aleatorias
- Programa principal
- Generador de informes

FLUJO DE CONTROL



¿Qué es un algoritmo?:

Es una fórmula para resolver un problema. Es un conjunto de acciones o secuencia de operaciones que ejecutadas en un determinado orden resuelven el problema. Existe n algoritmos, hay que coger el más efectivo.

Características:

- Tiene que ser preciso.
- Tiene que estar bien definido.
- Tiene que ser finito.

La programación es adaptar el algoritmo al ordenador.

El algoritmo es independiente según donde lo implemente.

EL LENGUAJE DE PROGRAMACIÓN:

Existen diferentes tipos, de bajo nivel y de alto nivel.

Instrucciones en una computadora y sus tipos:

Una instrucción es cada paso de un algoritmo, pero que lo ejecuta el ordenador. Un programa es un conjunto de instrucciones que ejecutadas ordenadamente resuelven un problema.

ALGORITMOS

El algoritmo de Simulación Monte Carlo Crudo o Puro está fundamentado en la generación de números aleatorios por el método de Transformación Inversa, el cual se basa en las distribuciones acumuladas de frecuencias:

- Determinar la/s V.A. y sus distribuciones acumuladas(F)
- Generar un número aleatorio uniforme Î (0,1).
- Determinar el valor de la V.A. para el número aleatorio generado de acuerdo a clases que tengamos.
- Calcular media, desviación estándar error y realizar el histograma**.**
- Analizar resultados para distintos tamaños de muestra**.**

Otra opción para trabajar con Monte Carlo, cuando la variable aleatoria no es directamente el resultado de la simulación o tenemos relaciones entre variables es la siguiente:

- Diseñar el modelo lógico de decisión

Especificar distribuciones de probabilidad para las variables aleatorias relevantes

Incluir posibles dependencias entre variables.

Muestrear valores de las variables aleatorias.

Calcular el resultado del modelo según los valores del muestreo (iteración) y registrar el resultado

Repetir el proceso hasta tener una muestra estadísticamente representativa

Obtener la distribución de frecuencias del resultado de las iteraciones

Calcular media, desvío.

Analizar los resultados

Las principales características a tener en cuenta para la implementación o utilización del algoritmo son:

- El sistema debe ser descripto por 1 o más funciones de distribución de probabilidad
- Generador de números aleatorios: como se generan los números aleatorios es importante para evitar que se produzca correlación entre los valores muéstrales.
- Establecer límites y reglas de muestreo para las fdp: conocemos que valores pueden adoptar las variables.
- Definir Scoring: Cuando un valor aleatorio tiene o no sentido para el modelo a simular.
- Estimación Error: Con que error trabajamos, cuanto error podemos aceptar para que una corrida sea válida
- Técnicas de reducción de varianza.
- Paralelización y vectorización: En aplicaciones con muchas variables se estudia trabajar con varios procesadores paralelos para realizar la simulación.

Ejemplos del funcionamiento de un programa con algoritmos:

AnyLogic

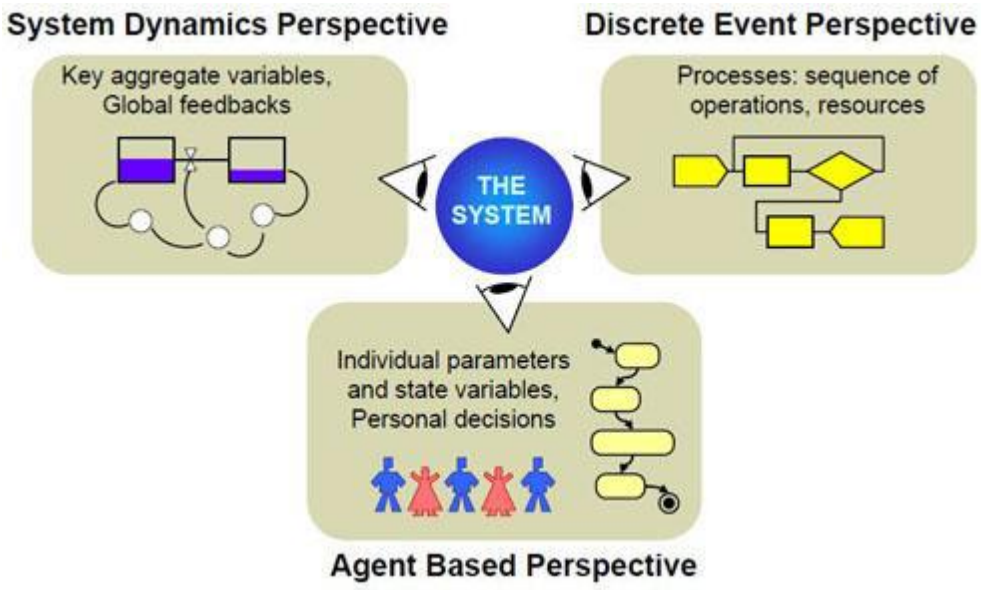
Anylogic es una herramienta desarrollada por la empresa rusa XJ tecnologías. Es una herramienta de simulación novedosa construida utilizando la ciencia más avanzada en la construcción de modelos y tecnologías de la información desarrollada en los últimos 10 años. Comparado con otras herramientas tradicionales, esta nos permite retornos más altos en esfuerzos de modelaje, porque puede:
Crear modelos mucho más rápidos con objetos visuales, flexibles, extensibles y reutilizables tanto estandarizados como personalizados y herramientas de java.
Modelar de una manera mucho más precisa y capturar más fenómenos al utilizar muchos ambientes de modelajes como: simulación de agentes, dinámica de sistemas, sistemas dinámicos, simulación continua y discreta. Es posible combinarlos, ajustarlos para afrontar un problema más específico.
Utilizar un conjunto de herramientas de optimización y análisis directamente del ambiente de modelaje que se esté utilizando.

Integrar de una manera fácil y eficiente la arquitectura abierta de los modelos de Anylogic con los software de oficina o corporativos, incluyendo hojas de cálculo, bases de datos, sistemas ERP y CRM. También se puede conectar el modelo en tiempo real con ambiente o sistema operacional.
Prolongar el ciclo de vida del modelo al actualizar el modelo a medida que el sistema del mundo real evoluciona.

AnyLogic Professional Edition es la solución definitiva para el desarrollo de modelos de simulación de grandes y complejas y sofisticadas animaciones, integrando modelos en varios entornos de TI, y cómo crear y utilizar bibliotecas personalizadas para áreas específicas de aplicación.
La terminología de simulación usado por AnyLogic consta de los siguientes elementos:

- **Diagramas de Depósitos** y flujo se utilizan para el modelado de Dinámica de Sistemas.
- **Diagramas de Estado** se utilizan principalmente modelando agentes para definir su comportamiento. También se utiliza a menudo en modelos de evento discreto, por ejemplo, para simular el fallo de una máquina.
- **Diagramas de Acción** se utilizan para definir los algoritmos. Pueden ser utilizados en el modelado de Eventos Discretos, por ejemplo, para el enrutamiento de llamadas, o el modelado basado en el agente, por ejemplo, para formular la lógica de decisiones usadas por el agente.
- **Organigramas** (Diagramas del Flujo de Processo) son las construcciones básicas utilizadas para definir el proceso de modelado de Eventos Discretos.

El idioma también incluye: construcciones de modelado al nivel de apoyo(variables, ecuaciones, parámetros, eventos, etc), formas de presentación (líneas, poli líneas, óvalos, etc), elementos de análisis (conjuntos de datos, histogramas, diagramas), herramientas de conectividad, las imágenes estándar, y estructuras experimentales.



Castalia

Castalia es un simulador de redes inalámbricas de sensores (WSN), redes de área corporal (BAN) y, en general las redes de dispositivos integrados de baja potencia. Se basa en la plataforma de la OMNeT ++ y puede ser utilizado por los investigadores y desarrolladores que quieran probar sus algoritmos y / o protocolos distribuidos en modelos de canal y de radio inalámbricos realistas, con un comportamiento realista nodo especial en relación con el acceso de la radio. Castalia también se puede utilizar para evaluar las diferentes características de la plataforma para aplicaciones específicas, ya que es altamente paramétrico, y puede simular una amplia gama de plataformas.
Las principales características de Castalia son:

- Modelo de canal avanzado basado en datos medidos empíricamente
- Modelo define un mapa de pérdida de trayectoria, no simplemente las conexiones entre los nodos
- Modelo complejo de la variación temporal de la pérdida de trayectoria
- Totalmente compatible con la movilidad de los nodos
- La interferencia se maneja como intensidad de la señal recibida, no como característica separada
- Modelo de radio avanzado basado en radios reales para la comunicación de baja potencia
- Probabilidad de recepción basado en SINR, tamaño de paquete, tipo de modulación. PSK FSK compatible, la modulación de encargo permitido por la definición de la curva de SNR-BER
- Múltiples niveles de potencia TX con variaciones de nodos individuales permitidos
- Los estados con el consumo de diferentes potencias y retardos de conexión entre ellos
- Detección de portadora flexible (sondeo-basado e interrumpir-based)
- Disposiciones de modelado de detección ampliado
- Modelo altamente flexible proceso físico
- Sintiendo el ruido del dispositivo, los prejuicios y el consumo de energía
- Sincronización del reloj de nodo, el consumo de energía de la CPU.
- Protocolos MAC y enrutamiento disponibles
- Diseñado para la adaptación y ampliación.

Plant Simulation

Tecnomatix Plant Simulation es una herramienta de simulación de eventos discretos que ayuda a crear modelos digitales de sistemas de logística (por ejemplo, producción), para permitir la exploración de las características de los sistemas y la optimización de su rendimiento. Mediante estos modelos digitales es posible llevar a cabo experimentos y trabajar con escenarios hipotéticos sin afectar a los sistemas de producción existentes, o bien (cuando se usan en el proceso de planificación) mucho antes de instalar los sistemas de producción en sí. Las completas herramientas de análisis, tales como análisis de cuellos de botella, estadísticas y diagramas, permiten evaluar distintos escenarios de

manufactura. Los resultados proporcionan la información necesaria para tomar decisiones fiables con rapidez, en las primeras fases de la planificación de producción.
Plant Simulation permite modelar y simular los sistemas de producción y sus procesos. Además, mediante la simulación en planta, se puede optimizar el flujo de materiales, la utilización de recursos y la logística en todos los niveles de planificación de planta, desde las instalaciones de producción global a las plantas locales o líneas específicas.

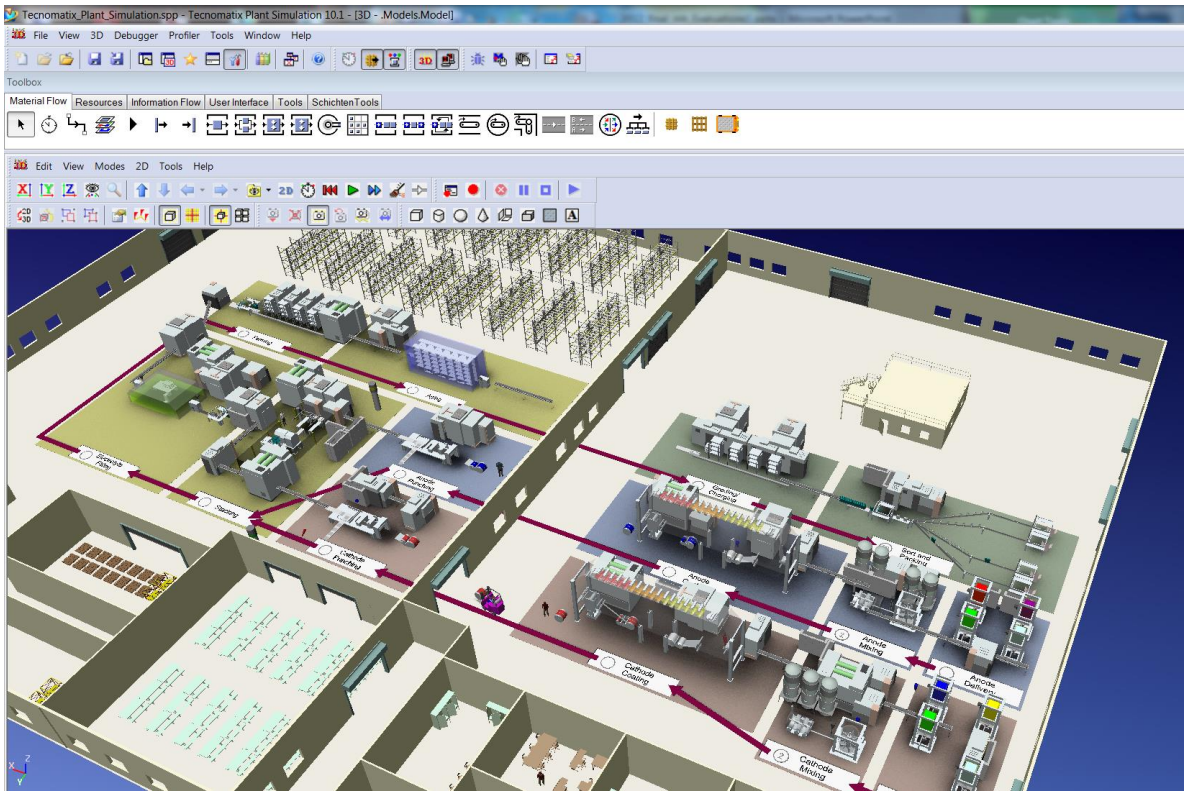
Capacidades

- Modelos orientados a objetos con estructura jerárquica
- Arquitectura abierta con varias interfaces estándar
- Administración de bibliotecas y objetos
- Optimización basada en algoritmo genético
- Análisis automático de los resultados de simulación
- Generador de informes basado en HTML

Aplicación

Plant Simulation es utilizado en la mayoría de las industrias. Especialmente en:

- Industria automotriz
- Proveedores automotrices
- Aeroespacial
- Plantas manufactureras
- Ingeniería mecánica
- Industria de procesos
- Industria de electrónicos
- Industria de productos de consumo
- Aeropuertos
- Compañías logísticas (logísticas de transportación, logísticas de transporte y logísticas de producción)
- Proveedores de almacenes altos, proveedores de vehículos guiados automáticamente y sistemas de monorriel eléctricos.
- Casas de consultoría y proveedores de servicios.



VISSIM

VISSIM es una herramienta de software para la simulación microscópica y multimodal del tránsito, desarrollada por la empresa PTV -- Planung Transport Verkehr AG en Karlsruhe, Alemania. El acrónimo deriva del alemán “Verkehr In Städten - SIMulation” (en castellano “Simulación de tránsito en ciudades"). El fundamento teórico de VISSIM se sitúa en la universidad de Karlsruhe en los años 80 y su primera aparición como herramienta comercial en entorno Windows de Microsoft fue en el año 1992, con la versión 2.03. Actualmente lidera el mercado mundial.

Simulación microscópica

En un modelo de “simulación microscópica” o modelo de microsimulación los individuos que componen los flujos de tránsito (vehículos, bicicletas, peatones, etc.) son el elemento mínimo. Sus características (físicas y psicológicas) y su interacción mutua y con elementos viales son modelizadas con reglas, algoritmos y modelos de comportamiento. Un modelo de microsimulación del tránsito es dinámico (que evoluciona en el tiempo), discreto (el estado de las variables cambia instantáneamente en tiempos puntuales, normalmente fijos) y estocástico (con resultados aleatorios).

Simulación multimodal

La "Simulación multimodal" se particulariza por modelar más de un tipo de tránsito y las interacciones entre éstos. En VISSIM pueden simularse los siguientes tipos de tránsito:

- Vehículos (coches, buses, camiones, motocicletas, etc.)
- Transporte público (tranvías, buses, etc.)
- Bicicletas
- Peatones
- Rickshaws

Aplicación

El ámbito de aplicación de VISSIM comprende desde la ingeniería del tránsito (sincronización y planificación de planes semafóricos, experimentación con sistemas inteligentes de transporte y sistemas de control y gestión del tránsito) pasando por la planificación del transporte, estudios de movilidad hasta visualizaciones en 3D para documentación ilustrativa y presentaciones.

4.3 Programacion Declarativa imperativa, Estructurada y Objeto

Programación Declarativa

Es un paradigma de programación basado en la lógica en el que se estudian de forma simple muchos aspectos avanzados de los lenguajes de programación modernos.

Características

La Programación Declarativa, en contraposición a la Programación Imperativa es un paradigma de programación que está basado en el desarrollo de programas especificando o "declarando" un conjunto de condiciones, proposiciones, afirmaciones, restricciones, ecuaciones o transformaciones que describen el problema y detallan su solución.

La solución es obtenida mediante mecanismos internos de control, sin especificar exactamente cómo encontrarla (tan sólo se le indica a la computadora que es lo que se desea obtener o que es lo que se está buscando). No existen asignaciones destructivas, y las variables son utilizadas con transparencia referencial

Aunque en la Programación Declarativa cabe más de un paradigma de programación, se ha optado por centrarla en el estudio de la denominada Programación Lógica, el cual está basado en el cálculo de proposiciones y sus relaciones lógicas.

La programación declarativa es un estilo de programación en el que el programador especifica qué debe computarse más bien que cómo deben realizarse los cómputos.

- "programa = lógica + control" (Kowalski)
- "algoritmos + estructuras de datos = programas" (Wirth)

El componente lógico determina el significado del programa mientras que el componente de control solamente afecta a su eficiencia.

La tarea de programar consiste en centrar la atención en la lógica dejando de lado el control, que se asume automático, al sistema.

La característica fundamental de la programación declarativa es el uso de la lógica como lenguaje de programación:

- Un programa es una teoría formal en una cierta lógica, esto es, un conjunto de fórmulas lógicas que resultan ser la especificación del problema que se pretende resolver, y

la computación se entiende como una forma de inferencia o deducción en dicha lógica. Los principales requisitos que debe cumplir la lógica empleada son:

- Disponer de un lenguaje que sea suficientemente expresivo.
- Disponer de una semántica operacional (un mecanismo de cómputo que permita ejecutar los programas).
- Disponer de una semántica declarativa que permita dar un significado a los programas de forma independiente a su posible ejecución.

Algunos lenguajes declarativos

Lenguajes lógicos

- Prolog (Programación funcional),
- ML (Programación funcional),
- Lisp (Programación funcional),
- Curry (Programación Lógico-Funcional)
- F-Prolog (Programación Lógica Difusa)

Programación Imperativa

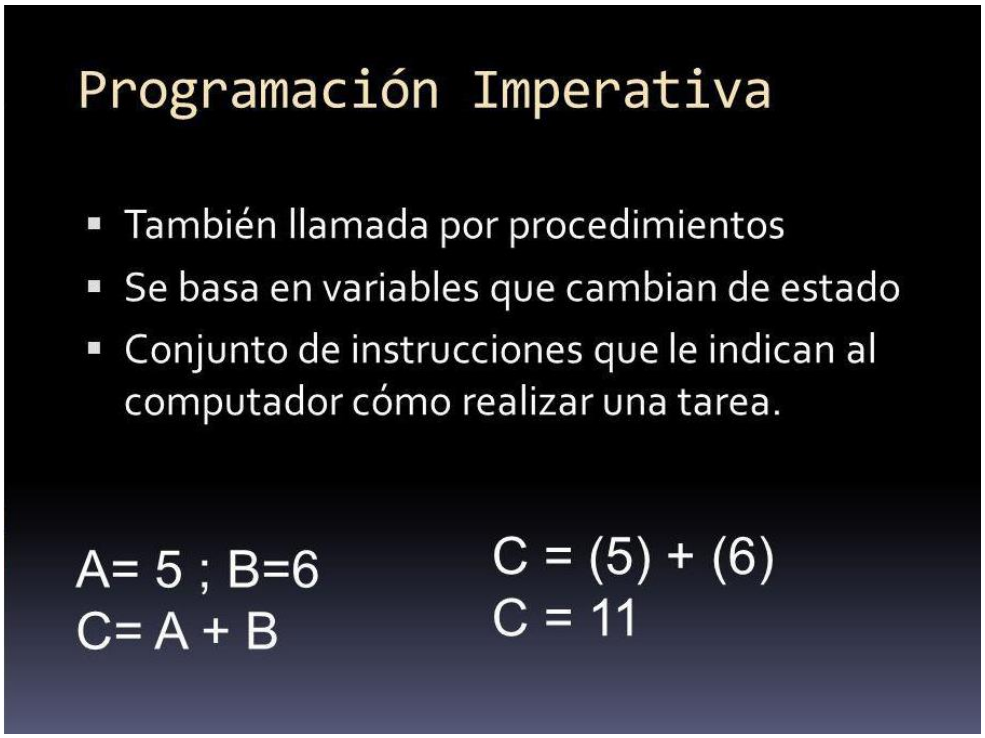
La programación imperativa, en contraposición a la programación declarativa es un paradigma de programación que describe la programación en términos del estado del programa y sentencias que cambian dicho estado. Los programas imperativos son un conjunto de instrucciones que le indican al computador cómo realizar una tarea.

La implementación de hardware de la mayoría de computadores es imperativa; prácticamente todo el hardware de los computadores está diseñado para ejecutar código de máquina, que es nativo al computador, escrito en una forma imperativa. Esto se debe a que el hardware de los computadores implementa el paradigma de las Máquinas de Turing. Desde esta perspectiva de bajo nivel, el estilo del programa está definido por los contenidos de la memoria, y las sentencias son instrucciones en el lenguaje de máquina nativo del computador (por ejemplo el lenguaje ensamblador).

Los lenguajes imperativos de alto nivel usan variables y sentencias más complejas, pero aún siguen el mismo paradigma. Las recetas y las listas de revisión de procesos, a pesar de no ser programas de computadora, son también conceptos familiares similares en estilo a la programación imperativa; cada paso es una instrucción, y el mundo físico guarda el estado (Zoom).

Los primeros lenguajes imperativos fueron los lenguajes de máquina de los computadores originales. En estos lenguajes, las instrucciones fueron muy simples, lo cual hizo la implementación de hardware fácil, pero obstruyendo la creación de programas complejos. Fortran, cuyo desarrollo fue iniciado en 1954 por John Backus en IBM, fue el primer gran lenguaje de programación en superar los obstáculos presentados por el código de máquina en la creación de programas complejos.

Las acciones principales que componen este paradigma son la asignación, selección e iteración.



Diferencias entre imperativo y declarativo

En la Programación Imperativa se describe paso a paso un conjunto de instrucciones que deben ejecutarse para variar el estado del programa y hallar la solución, es decir, un algoritmo en el que se describen los pasos necesarios para solucionar el problema.

En la programación declarativa las sentencias que se utilizan lo que hacen es describir el problema que se quiere solucionar, pero no las instrucciones necesarias para solucionarlo. Esto último se realizará mediante mecanismos internos de inferencia de información a partir de la descripción realizada.

Este estilo de programación encuentra numerosas aplicaciones industriales en campos como las bases de datos, ingeniería del software, procesadores de lenguajes, lenguaje natural, investigación operativa, seguridad de redes, etc.

Ventajas

Se ha dicho que los lenguajes declarativos tienen la ventaja de ser razonados matemáticamente, lo que permite el uso de mecanismos matemáticos para optimizar el rendimiento de los programas.

- Elegancia, claridad, sencillez, potencia y concisión.
- Semánticas claras, simples y matemáticamente bien fundadas.
- Cercanos al nivel de abstracción de las especificaciones formales/informales de los problemas a resolver.
- Referencialmente transparentes: Comportamiento matemático adecuado que permite razonar sobre los programas.
- Soportan técnicas muy avanzadas de desarrollo, mantenimiento y validación de programas.
- Altas dosis de paralelismo implícito.
- Aplicaciones variadas y de gran interés.
- Son fiables, elegantes y expresivos.

Resultados de corrección y completitud

Según la clase de lógica que se emplee como fundamento del lenguaje declarativo se obtendrá los diferentes estilos de programación declarativa.

- Ecuacional Funcional
- Clausal Relacional
- Heterogenea Tipos
- Géneros ordenados Herencia
- Modal S.B.C.
- Temporal Concurrencia

Programación estructurada

Programación Estructurada es una técnica en la cual la estructura de un programa, esto es, la interpelación de sus partes realiza tan claramente cómo es posible mediante el uso de tres estructuras lógicas de control:

a. Secuencia: Sucesión simple de dos o más operaciones.
b. Selección: bifurcación condicional de una o más operaciones.
c. Interacción: Repetición de una operación mientras se cumple una condición.

Estos tres tipos de estructuras lógicas de control pueden ser combinados para producir programas que manejen cualquier tarea de procesamiento de información.

Un programa estructurado está compuesto de segmentos, los cuales puedan estar constituidos por unas pocas instrucciones o por una página o más de codificación. Cada segmento tiene solamente una entrada y una salida, estos segmentos, asumiendo que no poseen lazos infinitos y no tienen instrucciones que jamás se ejecuten, se denominan programas propios. Cuando varios programas propios se combinan utilizando las tres estructuras básicas de control mencionadas anteriormente, el resultado es también un programa propio.

La programación Estructurada está basada en el Teorema de la Estructura, el cual establece que cualquier programa propio (un programa con una entrada y una salida exclusivamente) es equivalente a un programa que contiene solamente las estructuras lógicas mencionadas anteriormente.

Una característica importante en un programa estructurado es que puede ser leído en secuencia, desde el comienzo hasta el final sin perder la continuidad de la tarea que cumple el programa, lo contrario de lo que ocurre con otros estilos de programación. Esto es importante debido a que, es mucho más fácil comprender completamente el trabajo que realiza una función determinada, si todas las instrucciones que influyen en su acción están físicamente contiguas y encerradas por un bloque. La facilidad de lectura, de comienzo a fin, es una consecuencia de utilizar solamente tres estructuras de control y de eliminar la instrucción de desvío de flujo de control, excepto en circunstancias muy especiales tales como la simulación de una estructura lógica de control en un lenguaje de programación que no la posea.

VENTAJAS POTENCIALES

Un programa escrito de acuerdo a estos principios no solamente tendrá una estructura, sino también una excelente presentación.

Un programa escrito de esta forma tiende a ser mucho más fácil de comprender que programas escritos en otros estilos.

La facilidad de comprensión del contenido de un programa puede facilitar el chequeo de la codificación y reducir el tiempo de prueba y depuración de programas. Esto ultimo es cierto parcialmente, debido a que la programación estructurada concentra los errores en uno de los factores más generador de fallas en programación: la lógica.

Un programa que es fácil para leer y el cual esta compuesto de segmentos bien definidos tiende a ser simple, rápido y menos expuesto a mantenimiento. Estos beneficios derivan en parte del hecho que, aunque el programa tenga una extensión significativa, en documentación tiende siempre a estar al día, esto no suele suceder con los métodos convencionales de programación.

La programación estructurada ofrece estos beneficios, pero no se la debe considerar como una panacea ya que el desarrollo de programas es, principalmente, una tarea de dedicación, esfuerzo y creatividad.

TEOREMA DE LA ESTRUCTURA

El teorema de la estructura establece que un programa propio puede ser escrito utilizando solamente las siguientes estructuras lógicas de control: secuencia, selección e iteración.

Un programa de define como propio si cumple con los dos requerimientos siguientes:

a. Tiene exactamente una entrada y una salida para control del programa.
b. Existen caminos continuos desde la entrada hasta la salida que conducen por cada parte del programa, es decir, no existen lazos infinitos ni instrucciones que no se ejecutan.

Las **tres** estructuras lógicas de control básicas, se definen de la siguiente forma:

Secuencia: es simplemente la formalización de la idea de que las instrucciones de un programa son ejecutadas en el mismo orden en que ellas aparecen en el programa. En términos de diagrama de flujo la secuencia es representada por una función después de la otra, como se muestra a continuación.

A y B pueden ser instrucciones sencillas hasta módulos completos, lo importante es que sean programas propios, independientemente de su tamaño o complejidad interna. Ay B deben ser programas propios en el sentido en que estos fueron definidos, es decir, que posean solamente una entrada y una salida; la combinación de A seguida por B es también un programa propio, ya que esta unión tiene una entrada y una salida exclusivamente, esto se muestra gráficamente en la figura siguiente:

Donde la caja externa sugiere que la combinación de A seguida de puede ser tratada como una unidad para propósitos de control.

Selección: Es la escogencia entre dos acciones tomando la decisión en base al resultado de evaluar un predicado. Esta estructura de control es denominada usualmente IF THEN ELSE. La representación en forma de diagrama de flujo de esta estructura lógica de control se muestra a continuación:

F

Donde P es predicado y A y B son las dos funciones.

Iteración: Esta estructura lógica es utilizada para que se repita la ejecución de un conjunto de instrucciones mientras se cumpla una condición o predicado. Generalmente a esta estructura se le conoce como DOWHILE (hacer mientras) y su representación se muestra a continuación:

V

F

donde P es predicado y A es el modulo controlado.

Se debe comprender claramente que un rectángulo, que representa un modulo en un diagrama, siempre puede ser sustituido por cualquiera de las tres estructuras de control descritas anteriormente; por ejemplo, veamos el diagrama siguiente:

En él, la línea punteada limita un rectángulo que contiene una estructura, que a su vez controla dos módulos X y Y. La estructura limitada por la línea punteada es sustituida por una función quedando de la siguiente forma:

V

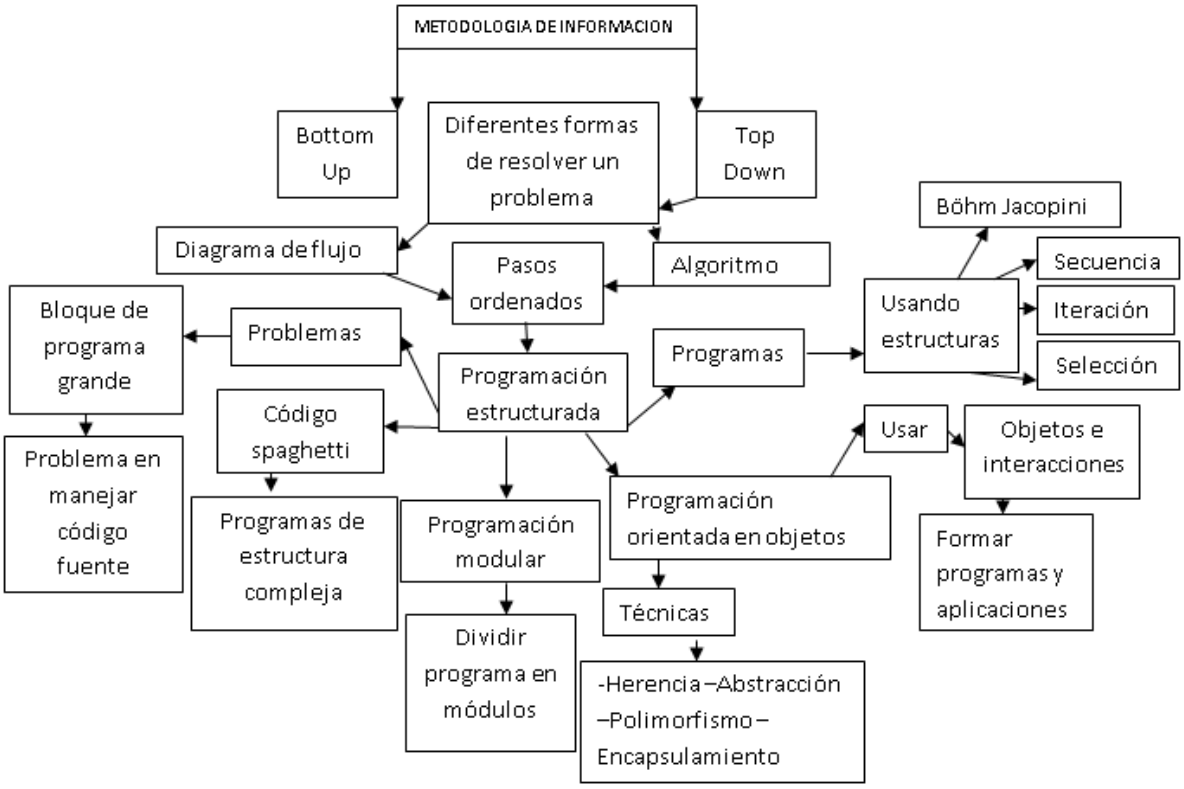
F

es decir, una función sustituye a una estructura lógica de control o viceversa.

VENTAJAS DE LA PROGRAMACION ESTRUCTURADA

Con la programación estructurada elaborar programas de computador sigue siendo un albor que demanda esfuerzo, creatividad, habilidad y cuidado. Sin embargo, con este nuevo estilo podemos obtener las siguientes ventajas:

1. - Los programas son más fáciles de entender. Un programa estructurado puede ser leído en secuencia, de arriba hacia abajo, sin necesidad de estar saltando de un sitio a otro en la lógica, lo cual es típico de otros estilos de programación. La estructura del programa es más clara puesto que las instrucciones están más ligadas o relacionadas entre sí, por lo que es más fácil comprender lo que hace cada función.
2. Reducción del esfuerzo en las pruebas. El programa se puede tener listo para producción normal en un tiempo menor del tradicional; por otro lado, el seguimiento de las fallas ("debugging") se facilita debido a la lógica más visible, de tal forma que los errores se pueden detectar y corregir más fácilmente.
3. Reducción de los costos de mantenimiento.
4. Programas más sencillos y más rápidos
5. Aumento de la productividad del programador
6. Se facilita la utilización de las otras técnicas para el mejoramiento de la productividad en programación
7. Los programas quedan mejor documentados internamente.



4.4 Ciclo de Vida de un Software

El término **ciclo de vida del software** describe el desarrollo de software, desde la fase inicial hasta la fase final. El propósito de este programa es definir las distintas fases intermedias que se requieren para validar el desarrollo de la aplicación, es decir, para garantizar que el software cumpla los requisitos para la aplicación y verificación de los procedimientos de desarrollo: se asegura de que los métodos utilizados son apropiados.
Estos programas se originan en el hecho de que es muy costoso rectificar los errores que se detectan tarde dentro de la fase de implementación. El ciclo de vida permite que los errores se detecten lo antes posible y por lo tanto, permite a los desarrolladores concentrarse en la calidad del software, en los plazos de implementación y en los costos asociados.

El ciclo de vida básico de un software consta de los siguientes procedimientos:

- **Definición de objetivos:** definir el resultado del proyecto y su papel en la estrategia global.
- **Análisis de los requisitos y su viabilidad:** recopilar, examinar y formular los requisitos del cliente y examinar cualquier restricción que se pueda aplicar.
- **Diseño general:** requisitos generales de la arquitectura de la aplicación.
- **Diseño en detalle:** definición precisa de cada subconjunto de la aplicación.

- **Programación (programación e implementación):** es la implementación de un lenguaje de programación para crear las funciones definidas durante la etapa de diseño.
- **Prueba de unidad:** prueba individual de cada subconjunto de la aplicación para garantizar que se implementaron de acuerdo con las especificaciones.
- **Integración:** para garantizar que los diferentes módulos se integren con la aplicación. Éste es el propósito de la prueba de integración que está cuidadosamente documentada.
- **Prueba beta (o validación),** para garantizar que el software cumple con las especificaciones originales.
- **Documentación:** sirve para documentar información necesaria para los usuarios del software y para desarrollos futuros.
- **Implementación**
- **Mantenimiento:** para todos los procedimientos correctivos (mantenimiento correctivo) y las actualizaciones secundarias del software (mantenimiento continuo).

El orden y la presencia de cada uno de estos procedimientos en el ciclo de vida de una aplicación dependen del tipo de modelo de ciclo de vida acordado entre el cliente y el equipo de desarrolladores.

Modelos de ciclo de vida

Para facilitar una metodología común entre el cliente y la compañía de software, los modelos de ciclo de vida se han actualizado para reflejar las etapas de desarrollo involucradas y la documentación requerida, de manera que cada etapa se valide antes de continuar con la siguiente etapa. Al final de cada etapa se arreglan las revisiones de manera que (texto faltante).

Un modelo de ciclo de vida del software:

- Describe las fases principales de desarrollo de software.
- Define las fases primarias esperadas de ser ejecutadas durante esas fases.
- Ayuda a administrar el progreso del desarrollo, y
- Provee un espacio de trabajo para la definición de un detallado proceso de desarrollo de software.

Modelo Cascada

Este es el más básico de todos los modelos, y sirve como bloque de construcción para los demás modelos de ciclo de vida. La visión del modelo cascada del desarrollo de software es muy simple; dice que el desarrollo de software puede ser a través de una secuencia simple de fases. Cada fase tiene un conjunto de metas bien definidas, y las actividades

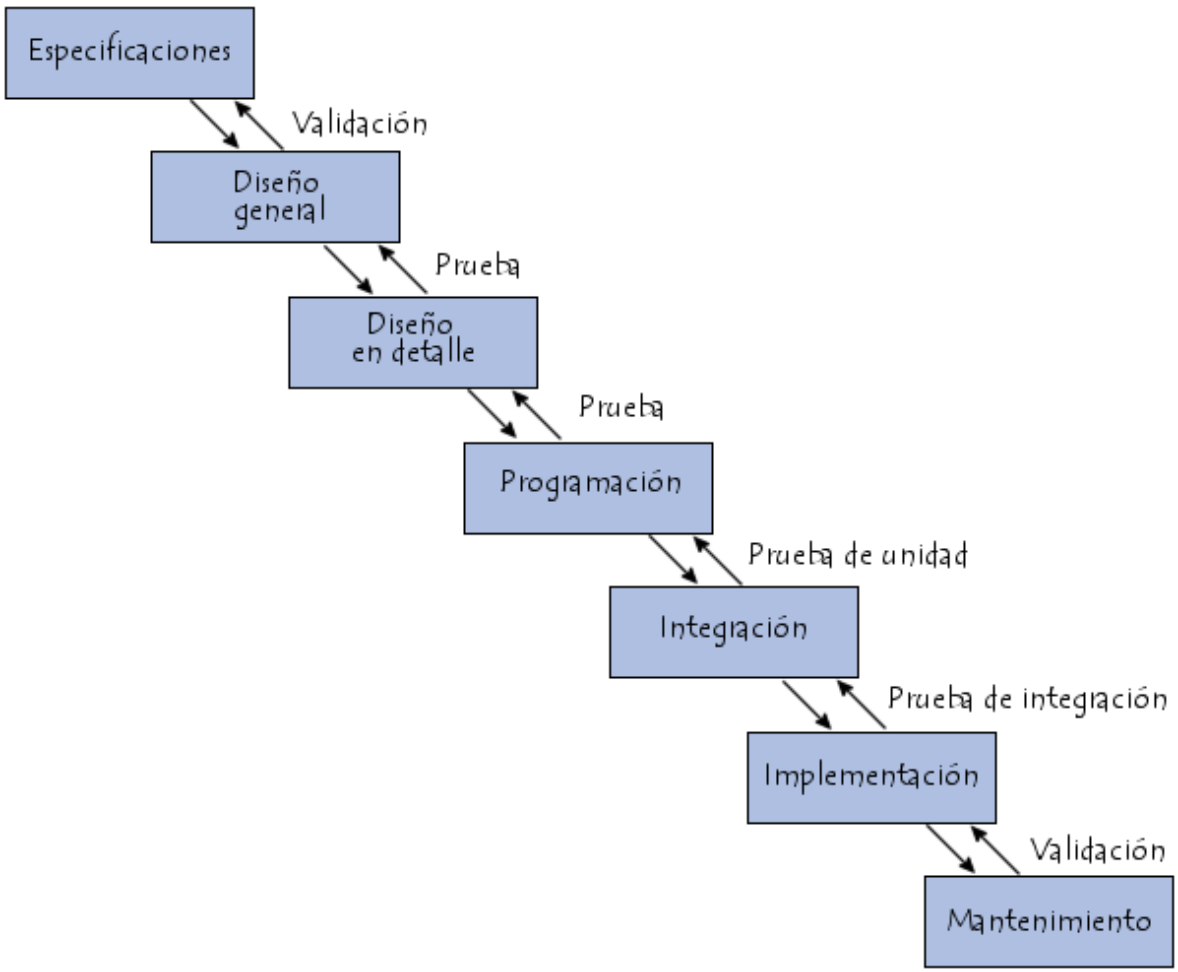
dentro de una fase contribuyen a la satisfacción de metas de esa fase o quizás a una subsecuencia de metas de la fase. Las flechas muestran el flujo de información entre las fases. La flecha de avance muestra el flujo normal. Las flechas hacia atrás representan la retroalimentación.

El modelo de ciclo de vida cascada, captura algunos principios básicos:

- Planear un proyecto antes de embarcarse en él.
- Definir el comportamiento externo deseado del sistema antes de diseñar su arquitectura interna.
- Documentar los resultados de cada actividad.
- Diseñar un sistema antes de codificarlo.
- Testear un sistema después de construirlo.

Una de las contribuciones más importantes del modelo cascada es para los administradores, posibilitándoles avanzar en el desarrollo, aunque en una escala muy bruta.

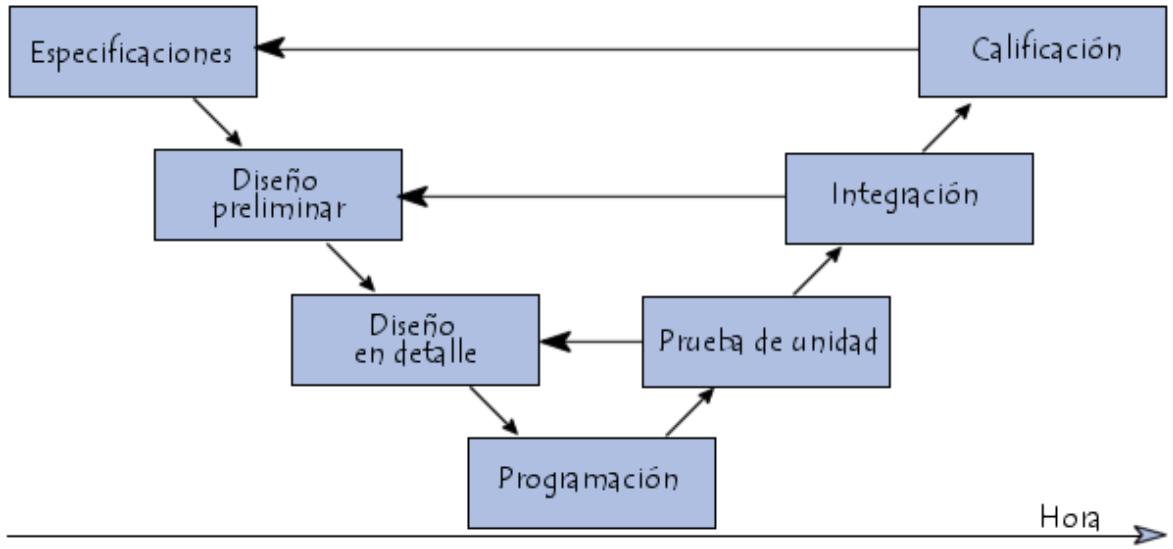
El modelo de ciclo de vida en cascada comenzó a diseñarse en 1966 y se terminó alrededor de 1970. Se define como una secuencia de fases en la que al final de cada una



de ellas se reúne la documentación para garantizar que cumple las especificaciones y los requisitos antes de pasar a la fase siguiente:

Modelo V

El modelo de ciclo de vida V proviene del principio que establece que los procedimientos utilizados para probar si la aplicación cumple las especificaciones ya deben haberse creado en la fase de diseño.



4.5 Ergonomía y Técnicas de Presentación de Resultado

Antes de adentrarnos a los aspectos ergonómicos de un software primero debemos de tener un concepto claro de lo que es la ergonomía. La Real Academia Española, define ergonomía como:

> "*Estudio de las condiciones de adaptación de un lugar de trabajo, una máquina, un vehículo; a las características físicas y psicológicas del trabajador o usuario.*"

Es decir, es una rama de saber que tiene como finalidad adaptar un entorno a las características o requerimientos del usuario.

Aplicándolo al diseño de software ya sea de simulación o con otros fines, hay que tener en mente tres puntos:

1.- Focalizan sus estudios en los aspectos físico y mental de los interfaces entre el usuario y los programas.

2.- Intentan diseñar procedimientos de diálogo y formatos de presentación que sean efectivos y fáciles de usar.

3.- Buscan que las aplicaciones sean fáciles de entender y aprender, y que potencien los conocimientos de quienes los usan.

Las razones principales por las cuales hay que tener en cuenta éste factor al momento de programar radican en tres aspectos primordiales, los cuales son:

1) Un software fácil de usar, fácil de aprender y entretenido en su manejo; reducen los costes de formación y el tiempo de aprendizaje a la vez que incrementan la productividad, el uso de los ordenadores y la motivación para trabajar confortablemente en el puesto de trabajo.

Un ejemplo sería cuando se compara una aplicación de ventanas, basada en un entorno Microsoft Windows, con la típica aplicación basada en el tratamiento de caracteres, se pueden llegar a reducir los períodos de aprendizaje hasta un 50 por ciento, dependiendo de la experiencia previa del usuario en entornos Windows. Para ciertas empresas, este ahorro puede significar mil

2) Los aspectos ergonómicos son hoy en día uno de los principales argumentos de venta. La principal diferencia entre distintos productos o casas de software conciernen a los aspectos ergonómicos.

3) Dentro de ISO, la norma 9241, partes 8 a 17, cubre también aspectos relacionados con el software y obliga a los desarrolladores a tener en cuenta sus indicaciones. Muchos países usarán las normas ISO para crear su propia legislación local acerca de los programas y aplicaciones software.

Las normas ISO aplicables al diseño ergonómico son varias y abarcan desde la realización de guías de usuario hasta la confección de los cuadros de diálogo, como más adelante se verá.

Aspectos ergonómicos a considerar

Cuando alguien está trabajando con un producto software, espera que la interacción o diálogo con el ordenador sea fácil y eficiente. Las técnicas de diálogo incluyen menús, comandos, manipulaciones directas y formatos de entrada de datos.

Un producto software que quiera ser catalogado como ergonómico, deberá seguir los siguientes siete principios que rigen el diseño de los diálogos ergonómicos:

1) Adecuado para el trabajo al que se destina: dotar a los productos de herramientas adecuadas, es decir, una clara definición de lo que realizan las diferentes opciones del software además de ocultar la complejidad de los procesos internos que el programa y/o el ordenador esté realizando.
2) Autodescriptivo: la aplicación debe de ser fácil y rápidamente comprendida por el usuario, en otras palabras el usuario no tendrá que verificar en el manual qué es lo que se puede esperar de un determinado menú o qué significado tienen los diferentes términos y palabras que aparezcan en pantalla. El sistema deberá ofrecer al usuario un sistema de diálogo claro, simple y conciso, apoyado por un mecanismo de pantallas de ayuda de fácil acceso, que contengan explicaciones concretas.
3) Controlable: el consumidor tendrá en todo momento la posibilidad de cancelar acciones que haya emprendido, deshacer los últimos comandos que haya ordenado y gobernar sus dispositivos de entrada y salida de datos.
4) Conforme a las expectaciones que genera: que cumpla con las cualidades con las que ha sido publicitado y que todos los componentes cumplan con la función que deben de desempeñar.
5) Tolerante con los errores que el usuario pueda cometer: es importante que no permita a los usuarios el ejecutar tareas que puedan provocar un error irrecuperable. No sólo deben detectar y avisar al usuario de los errores, sino que deben tratar de prevenir al mismo de lo que puede suceder. Cuando se produzca un error, el usuario podrá ser capaz de salir del mismo, comprender qué es lo que ha sucedido y tener a su elección una serie de opciones de salida del proceso.
6) Personalizable por el usuario: cualquier usuario independientemente de su nivel de conocimientos, podrá personalizar su área de trabajo según le convenga y según conciba que puede aumentar la efectividad del uso de los programas. Podrá definir colores, formas, agrupamiento de iconos, creación y agrupamiento de macros, entre muchas otras cualidades.
7) Documentado suficientemente para facilitar su aprendizaje: contarán con explicaciones coherentes (tanto en pantalla como en manuales) que vayan encaminadas a facilitar la labor de aprendizaje.

Técnicas de presentación de resultados

Una presentación es una forma de ofrecer y mostrar información de datos y resultados de un proceso. Con la presentación se dispone de un contenido multimedia (es decir cualquier apoyo visual o auditivo) que de una referencia sobre el tema y ayude a explicar los datos obtenidos en el proceso.
En lo que concierne al mundo de la simulación, la representación de resultados de manera general, puede ser de dos maneras gráfica o textual.
En un principio, la representación textual era la predilecta de los programas de simulación, consistía en la representación alfanumérica línea por línea de los resultados obtenidos durante la simulación. Éste suele ser cansado y tedioso, ya que la pantalla del ordenador se llena de líneas de números, dejándole todo el trabajo de interpretación y presentación al usuario.

No obstante, con el desarrollo de nueva tecnología se fue desarrollando poco a poco sistemas de gráficos que permitieron pasar de pantallas llenas de líneas con números a gráficas y animaciones; agilizando el análisis de los resultados obtenidos y reduciendo el tiempo empleado en su interpretación.

En la actualidad la representación de los datos admiten sistemas completamente gráficos o numéricos, sin embargo los más utilizados son los híbridos, donde tablas y gráficos se utilizan para la exposición de resultados.

En lo que corresponde a la representación textual, hay que tener en cuenta el tipo de variable que vayamos a representar si es cualitativa, cuantitativa, discreta o continua.

Si adoptamos un sistema gráfico, tenemos varias opciones para poder representar nuestros resultados, no obstante, por convención se utilizan principalmente cuatro tipos de gráficos:

1) Diagramas de líneas

Utilizados para la representación de relaciones entre variables, haciendo ahínco a las correspondencias de incremento o decremento de los datos.

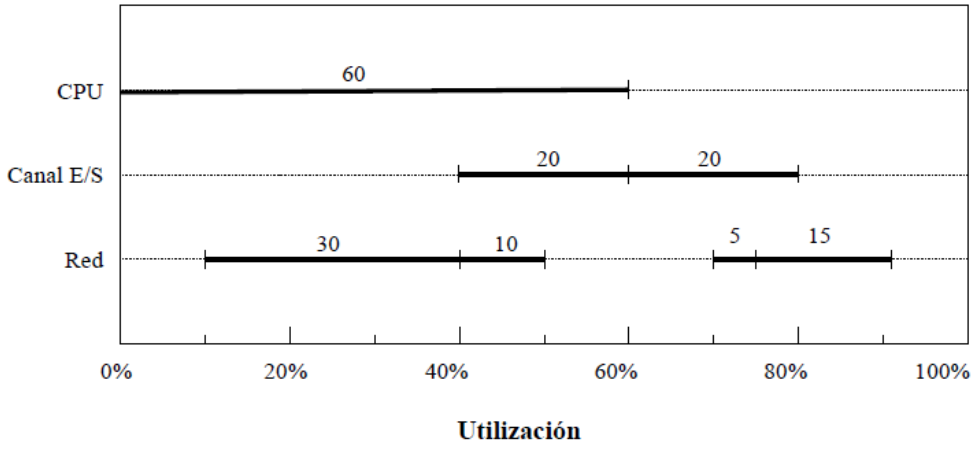
2) Diagramas de barras o histogramas

Se usan cuando lo que se quiere expresar es el incremento/decremento de varios recursos, donde un recurso no está estrechamente relacionado con el anterior o posterior, pero de una manera general si lo hacen. Por ejemplo, se puede relacionar los accidentes ocurridos al mes durante un año con un diagrama de barras; donde cada mes (recurso) estará relacionado con los otros porque pertenecen al mismo año (grupo de datos), más la incidencia de accidentes de cada uno, no tiene ninguna relación con la de los otros meses.

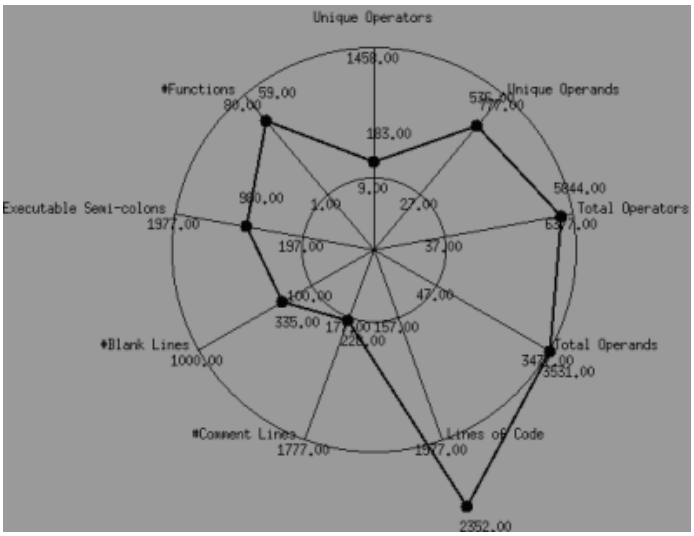
3) Diagramas de Gantt

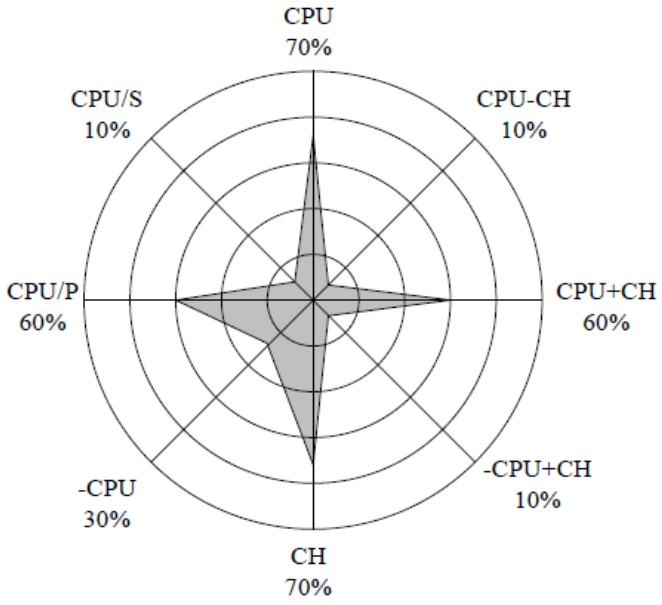
Muestra la duración relativa de condiciones o estados, partiendo de las primicias que un recurso con una utilización alta puede generar un cuello de botella y degradar el rendimiento y que uno con utilización baja representa una ineficiencia. Pretende la óptima utilización de recursos.

En ellos cada condición es representada por un conjunto de segmentos de línea, lo cual permite mostrar varios recursos; más **no** muestra dependencias.



4) ***Gráficas de Kiviat****:* Nos permite representar 3 o más índices de rendimiento, de una manera clara y precisa.





5 Diseño de un software de simulación

5.1 Estructura de un Programa

Encabezados

La primera está delimitada por la Cabecera del programa y por la palabra reservada BEGIN, y en ella se declaran o se definen todos los elementos habituales de programación (variables, subprogramas, etc.) que se van a utilizar en el programa y que están disponibles en el lenguaje.

La Zona de Instrucciones, delimitada por las palabras reservadas BEGIN y END, es la zona de las instrucciones ejecutables (las cuales utilizan los elementos declarados en la zona de declaraciones), es decir, la codificación del algoritmo que resuelve el problema para el que fue diseñado el programa.

La cabecera del programa consta de la palabra reservada PROGRAM seguida del Nombre del Programa y de punto y coma (;). El punto y coma es el separador de sentencias en Pas-cal,

Declaraciones

En todo programa de TurboPascal es necesario declarar o definir previamente todo lo que se vaya a utilizar y que no tenga un significado específico o *a priori* para este lenguaje de programación. En esta sección se realizan estas definiciones o declaraciones del programa.

Exceptuando la declaración de utilización de unidades que, si existe, es única y deberá incluirse al principio, el número y orden de las demás declaraciones no es rígido. A este respecto, la única norma general que es necesario respetar es que *cualquier elemento que se utilice en un punto determinado del programa deberá haber sido declarado previamente*.

Declaración de utilización de unidades

La sentencia de declaración de unidades especifica el nombre o identificador de las unidades que se van a utilizar en el programa. Como se verá más adelante detenidamente, una unidad es una colección de declaraciones de constantes, tipos de datos, variables, funciones y procedimientos que pueden emplearse en un programa de TurboPascal. Si son varias unidades se podrán declarar en la misma sentencia separándolas por comas.

Sintaxis: USES *Unidad1*, *Unidad2*, *Unidad_n*;

Si existe una sentencia de declaración de unidades en un programa deberá colocarse al principio de la sección de declaraciones de dicho programa, es decir, antes de cualquier otra declaración. En el siguiente ejemplo se declara el uso en el programa de dos unidades denominadas Crt y Dos:

Ej.: Uses Crt, Dos;

Declaración de etiquetas

Permiten realizar saltos incondicionales en la secuencia de instrucciones de un programa. Su utilización va unida a la sentencia goto y, aunque es un elemento incluido en la sintaxis de Pascal estándar, no se recomienda por la filosofía de la programación estructurada (que evita los saltos incondicionales).

Sintaxis: LABEL *Etiqueta1*, *Etiqueta2*, *Etiqueta_n*;

Una etiqueta es un *identificador* o una secuencia de cuatro dígitos decimales (entre 0 y 9999). Si las etiquetas son varias se podrán declarar en la misma sentencia separándolas por comas.

Ej.: Label 100, 200;

Declaración de constantes

Las constantes son datos que no cambian durante la ejecución del programa y que se definen durante el tiempo de compilación.

Sintaxis: CONST *Nombre_Constante = Expresion_1*;

Nombre_Constante_2 = Expresión_2;

Datos 23

Nombre_Constante_3 = Expresión_3;...

Si se declaran varias constantes en un programa podrán incluirse en una única sentencia CONST separando cada declaración de las demás con caracteres de punto y coma, aunque también puede haber varias sentencias CONST en la sección de declaraciones de un programa.

Ej.:

```
Const Pi = 3.1415;  {constante numérica real}
Limite = 325; {constante numérica  entera}
Saludo = '¡Hola!'; {cadena  de caracteres}
```

Declaración de tipos de dato

Un tipo de dato es un conjunto de valores de datos. En el lenguaje de programación TurboPascal todo dato ha de pertenecer a algún *tipo* determinado. Esta especificación determinará cómo se almacenará el dato correspondiente y qué operaciones se podrán realizar con dicho dato durante la ejecución del programa.

En TurboPascal hay tipos predefinidos que no es necesario declarar (tipos de datos numéricos enteros, numéricos reales, lógicos o booleanos, caracteres...) y otros que no lo están y que el programador deberá declarar.

La declaración de un tipo de dato consta del nombre o identificador del tipo de dato seguido de los valores que pueden tomar los datos de ese tipo. Por otro lado, existe la posibilidad de que algunos tipos puedan ser subconjuntos o *subrangos* de otros tipos. También es necesario declarar estos tipos de datos.

Definicion_n puede ser una lista de valores que van entre paréntesis (tipo de dato enumerado), un subconjunto de otro tipo ya definido o tipo subrango (en este caso se indica el valor inicial y final que define el subconjunto) o la especificación en cuanto a tamaño o estructura de un tipo de dato más complejo o *estructurado*.

Si se declaran varios tipos de dato en un programa podrán incluirse en una única sentencia TYPE separando cada declaración de las demás con caracteres de punto y coma. En cualquier caso, también puede haber varias sentencias TYPE en la sección de declaraciones de un programa.

En el siguiente ejemplo se incluyen en una misma declaración, los tres primeros son tipos de datos enumerados, los dos siguientes son de tipo subrango (numérico entero y de caracteres, respectivamente) y el último es de tipo estructurado como cadena de veinte caracteres.

```
Ej.: type palo = (bastos, oros, copas, espadas);
Estado = (soltero, casado, viudo);
dia = (lu,ma,mi,ju,vi,sa,dm);
digito = 0..9;
minuscula = 'a'..'z';
nombre = string[20];
```

Declaración de variables

Una variable es un espacio de la memoria reservado durante la ejecución del programa a

la que se le asocia un nombre o identificador y en la que se puede almacenar un valor que puede

cambiar durante dicha ejecución. La declaración consta de la palabra VAR seguida del

identificador de cada variable y su tipo, que puede ser predefinido o estar definido previamente

en la sección anterior.

Sintaxis: VAR *Variable*: *Tipo*;

Si se declaran varias variables del mismo tipo pueden incluirse en la misma sentencia de

declaración separadas por comas.

Fundamentos de programación - A. García-Beltrán, R. Martínez y J.A. Jaén 24

Sintaxis: VAR *Variable_1*,*Var_2*,...,*Var_n*: *Tipo*;

Si se declaran varias variables en un programa podrán incluirse en una única sentencia

VAR separando cada declaración de las demás con caracteres de punto y coma, aunque también

puede haber varias sentencias VAR en la sección de declaraciones de un programa.

Ej.:

```
VAR x,y,z : Real;
i,j : Integer;
condición : estado;
nota : digito;
libra : dia;
```

En el ejemplo anterior se declaran ocho variables de las cuales las cinco primeras son de

tipos predefinidos por TurboPascal (tres de tipo Real y dos de tipo Integer,

respectivamente) y las tres últimas aprovechan las declaraciones de tipos de dato del ejemplo de

apartado anterior.

Al declarar una variable se reserva espacio en memoria para almacenar los valores que

va tomando dicha variable durante la ejecución del programa. La cantidad de memoria

reservada dependerá del tipo de variable. Una variable de tipo Integer es una variable

numérica entera que ocupa 2 bytes (16 bits) de memoria, mientras que una de tipo Real, es una

variable numérica real que necesita 6 bytes (48 bits).

Declaración de funciones y procedimientos

Las funciones y procedimientos son las rutinas, subrutinas o subprogramas de Pascal.

Una rutina es un conjunto de instrucciones que pueden ejecutarse en cualquier lugar del programa principal o, dentro de otras subrutinas, sólo referenciando su nombre o identificador.

Como se verá más adelante, existen rutinas ya predefinidas o estándar en TurboPascal. Se tendrán que declarar obligatoriamente las subrutinas no predefinidas que vayan a utilizarse en el programa o que no estén incluidas en unidades cuyo uso se declare en el programa.

Las subrutinas tienen una estructura muy parecida a los programas con las excepciones de que su cabecera empieza por la palabra FUNCTION o PROCEDURE y su cuerpo no acaba en un punto sino en un carácter de punto y coma. Si bien tanto las funciones como los procedimientos pueden ejecutar una serie de sentencias, las funciones se diferencian de los procedimientos en que, una vez finalizada su ejecución, devuelven un valor, cuyo tipo de dato se especifica al final de la cabecera.

Partes de un programa

Introducción En este tema se presenta una introducción a Pascal: ¿qué es un lenguaje de programación?, un poco de historia sobre Pascal, ¿qué es compilar un programa, etc. Si no te interesa, puedes saltarlo ya que no es demasiado importante.	**Entrada y salida de datos** Aquí aprenderás a comunicarte con el usuario a través de tus programas. ¿Cómo? Pues a través de unos procedimientos que te permiten leer y escribir datos en la salida y entrada estándar respectivamente.
Estructura de un programa En este apartado podrás aprender de qué partes consta un programa en Pascal. Verás el orden en el que se escriben, en qué consisten, cuáles son obligatorias y cuáles no, cómo se relacionan unas con otras, etc.	**Sentencias y expresiones** En este tema se muestran los tipos de sentencias, de expresiones y de operadores que puedes utilizar en tus programas. Estos tres conceptos están relacionados, ya que los operadores se usan en expresiones, y éstas a su vez en sentencias.
Variables y constantes Si pinchas en este libro aprenderás cosas sobre las variables: qué es una variable, cómo se declara, cómo se inicia, etc. Además observarás que una variable pertenece a un tipo de dato, concepto éste que se trata en el siguiente tema.	**Control del flujo** En este tema aprenderás a que tus programas puedan variar el orden de su ejecución si se cumplen ciertas condiciones (sentencias selectivas). Y también aprenderás a repetir algo varias veces (sentencias iterativas).
Tipos de datos En este capítulo verás lo que es un tipo de dato. Este es un concepto muy importante en cualquier lenguaje de programación, especialmente en Pascal que es un lenguaje fuertemente tipeado.	**Programación modular** Aquí conocerás cómo construír pequeños programas (procedimientos y funciones) que ayuden a solucionar un problema grande dividiéndolo en subproblemas

5.2 Ejecución paralela: Threads y Fibers

En la ciencia de la computación, un thread de ejecución es la unidad más pequeña de proceso que un sistema operativo puede generar. Generalmente, resulta de hacer un fork de un programa en dos o más tareas corriendo concurrentemente. La implementación de threads y procesos es distinta en cada sistema operativo, pero, en la mayoría de los

casos, un thread está contenido dentro de un proceso. Pueden existir varios threads en un mismo proceso y pueden compartir recursos como la memoria, pero los distintos procesos no comparten estos recursos. En particular, los threads de un proceso comparten las instrucciones del proceso (su código) y su contexto (los valores de sus referencias en un cierto momento). Para ofrecer una analogía, varios threads en un proceso se parecen a varios cocineros leyendo el mismo libro de recetas y siguiendo las instrucciones, pero no necesariamente de la misma página.

En un solo procesador, el multithreading generalmente como una multiplicación de tiempo dividido (como en multitasking): el procesador va cambiando entre distintas threads. El context switch de este tipo sucede a tal velocidad que el usuario percibe que los threads o las tareas están corriendo al mismo tiempo. En un multiprocesador o sistema de más de un núcleo, de hecho, los threads o tareas van a correr al mismo tiempo, ya que cada procesador va a correr un thread en particular.

Muchos de los sistemas operativos modernos directamente tienen soporte para threading del tipo multiprocesador con un planificador de procesos. El kernel de un sistema operativo permite que los programadores puedan manipular threads mediante system calls. Se llama a algunas implementaciones de este thread, “kernel thread”. Un proceso liviano (LWP por sus siglas en ingles) es un tipo específico de kernel thread que comparte el mismo estado y la misma información.

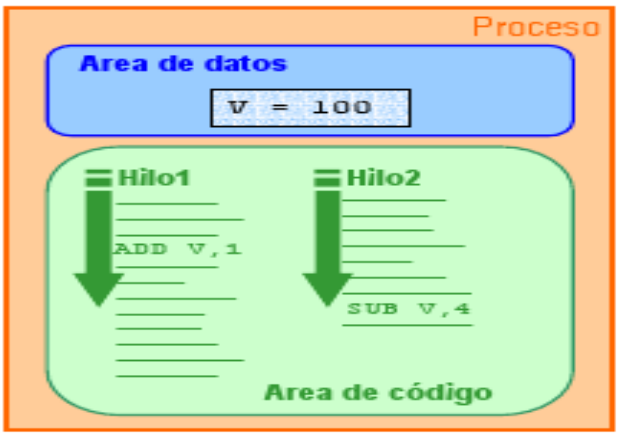
Un programa puede tener threads definidos por el usuario, por ejemplo, cuando se hace threading con timers o algún otro tipo de método que interrumpe su propia ejecución.

Un *thread* (LWP Light Weight Process) es flujo de ejecución de un proceso. Es la unidad básica de utilización del CPU y está constituida por:

- PC
- Conjunto de registros
- Espacio para pila

Los *threads* comparten con otros *threads*:

-- Código
- Datos
- Archivos abiertos, señales.



Ventajas y usos de Multithreading

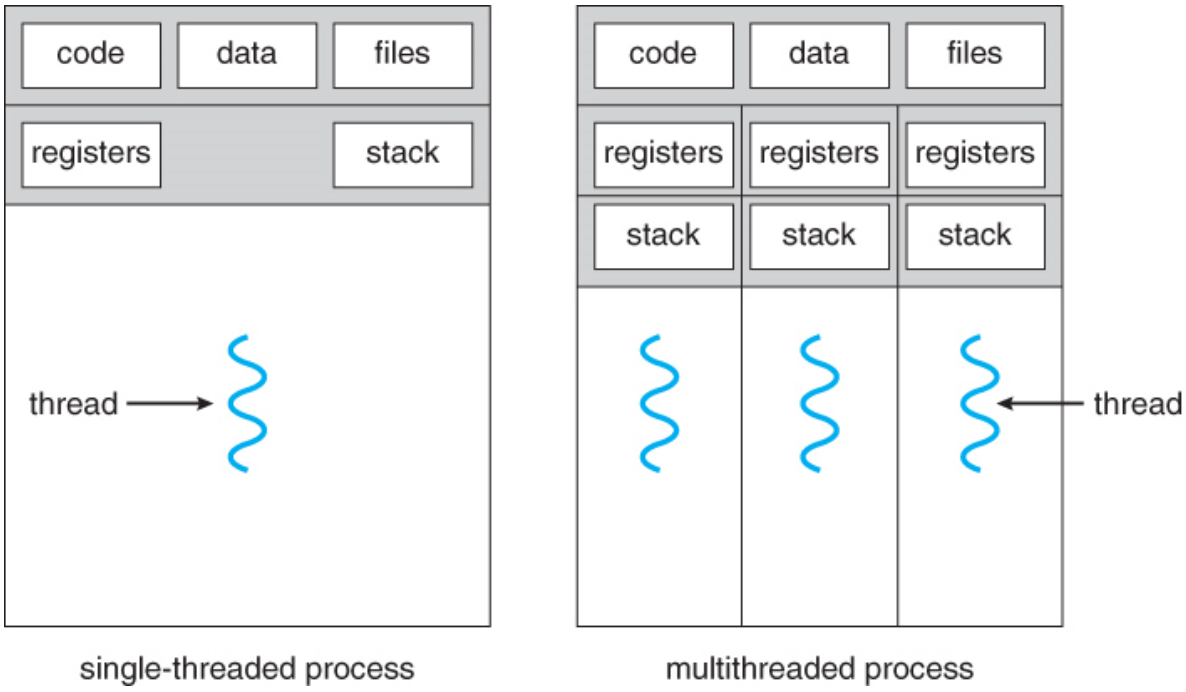
Multithreading, como un modelo extendido de programación y ejecución, permite que existan muchos threads dentro del contexto de un solo proceso. Estos threads tienen los mismos recursos que el proceso pero son capaces de ejecutarse independientemente. El modelo de programación con threads provee a los desarrolladores de una abstracción muy útil para lograr la ejecución concurrente. Sin embargo, quizás la aplicación más interesante de esta tecnología es cuando se aplica a un solo proceso para permitir una ejecución paralela en un sistema multiprocesador.

La ventaja antes mencionada de un programa con multithreading permite que opere más rápido en sistemas que tienen múltiples CPU, CPU con múltiples núcleos, o a través de un cluster, ya que los threads de un programa naturalmente tienden a tener ejecución concurrente verdadera. En tal caso, el programador tiene que tener cuidado y evitar condiciones de carrera, y otras conductas no intuitivas. Para manipular los datos de manera correcta, los threads van a necesitar, muchas veces, “planificaciones” o métodos para garantizar la llegada ordenada de estos threads, para poder procesar la data en el orden correcto. Los threads también pueden requerir operaciones mutex (que, en general, se implementan usando semáforos) para evitar que se modifiquen simultáneamente los datos compartidos, o que se lean mientras están siendo modificados. El no tener cuidado al usar tales primitivas, puede resultar en deadlocks.

Otra ventaja del multithreading, incluso para sistemas con una sola CPU, es la habilidad de una aplicación de mantenerse responsiva a input. En un programa de un solo thread, si el thread de ejecución principal se bloquea durante una tarea de larga duración, parecerá que se colgó toda la aplicación. Al mover este tipo de tareas de larga duración a un thread secundario, que corre concurrentemente con el thread de ejecución principal, es posible que la aplicación siga reconociendo el input del usuario mientras ejecuta tareas en el background.
Hay dos formas en que los sistemas operativos pueden generar threads:

1. Multithreading prioritario, se considera la forma superior de generar threads, ya que le permite al sistema operativo determinar cuando debe ocurrir un context switch. La desventaja de este método, es que el sistema quizás haga un context switch en un momento inapropiado, y así cause una inversión de prioridades o algún otro efecto negativo que puede evitarse usando la otra alternativa: multithreading cooperativo.

2. Multithreading cooperativo, en cambio, depende de que los threads mismos cedan el control cuando se detienen, algo que puede generar problemas si un thread está esperando a que un recurso esté disponible.



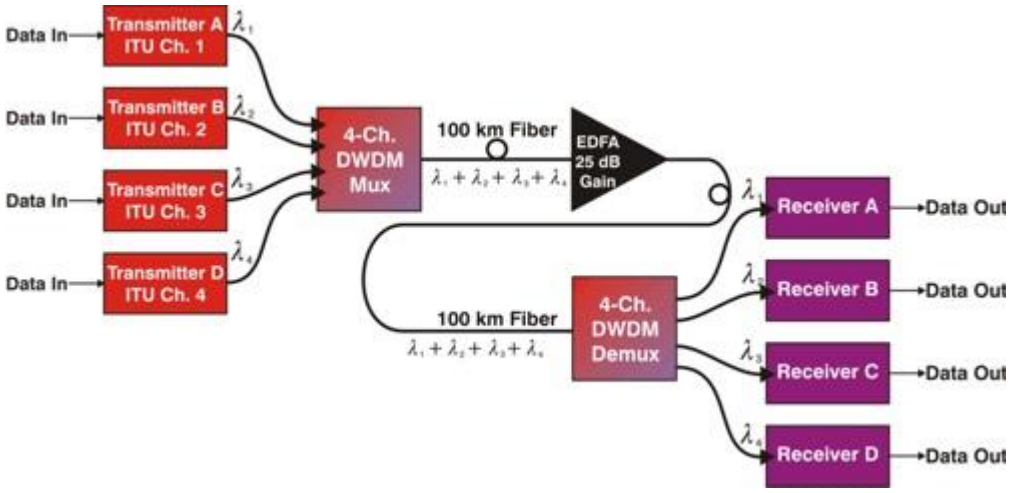
Procesos, kernel threads, threads de usuario y fibers

Un proceso es la unidad más pesada que puede planificar el kernel. Los procesos poseen recursos alocados por el sistema operativo. Los recursos incluyen memoria, file handles, sockets, device handles, y ventanas. Los procesos no comparten espacio de direcciones o recursos del archivo a menos que sea a través de métodos explícitos como heredar file handles o segmentos de memoria compartidos, o mappear el mismo archivo de manera compartida.
Un kernel thread es la unidad más liviana que puede planificar el kernel. Hay al menos un kernel thread en cada proceso. Si existen múltiples kernel threads dentro del mismo proceso, comparten el mismo espacio de memoria y recursos del archivo. Si el planificador de procesos del sistema operativo es prioritario, los kernel threads se van a planificar prioritariamente. Los kernel threads no tienen recursos salvo por una pila, una copia de registros que incluye el contador de programas y, en algunos casos, un almacenamiento de threads locales. El kernel es capaz de asignar un thread a cada núcleo lógico en un sistema (debido a que cada procesador se divide en múltiples núcleos lógicos si tiene soporte para multithreading, o si solo tiene suporte para un núcleo lógico por núcleo físico si no tiene soporte para multithreading) y puede reemplazar threads

bloqueados. Sin embargo, toma más tiempo reemplazar un kernel thread que un thread de usuario.
A veces, los threads se implementan en el espacio del usuario, y pasan a llamarse threads de usuario. El kernel no tiene noción de la existencia de los threads de usuario, y debido a eso se administran y generan en el espacio del usuario. Algunas implementaciones usan sus threads de usuario sobre varios kernel threads para beneficiarse en máquinas multiprocesadoras (modelos muchos a muchos). En este capitulo de la wiki, se usará el término “thread” para referirse a los kernel threads. Cuando los threads de usuarios son implementados por máquinas virtuales, se los suele llamar threads verdes. Generalmente, los threads de usuario son fáciles de crear y administrar, pero no pueden usar multithreading y multiprocesamiento (y pierden la ventaja que esto trae), y se bloquean si todos sus kernel threads asociados están bloqueados, incluso en el caso que haya otros user threads listos para ejecutarse.

Las fibers (fibras en castellano) son una unidad todavía más liviana de planificación que son planificadas de manera cooperativa: una fiber en ejecución debe explícitamente “ceder” para que otra fiber pueda ejecutarse, algo que hace su implementación más fácil que la de los kernel threads o user threads. Se puede generar una fiber para que se ejecute en cualquier thread en el mismo proceso. Esto le permite a las aplicaciones mejorar su rendimiento al administrar la planificación ellas mismas, en vez de depender de un planificador de kernel (que puede que no esté configurado especialmente para la aplicación).

Problemas con los threads y las fibers - Concurrencia y estructuras de datos
Los threads en un mismo proceso, comparten el mismo espacio de memoria. Esto permite que el código que se ejecute concurrentemente este fuertemente acoplado y que pueda también intercambiar datos sin el overhead o la complejidad que trae aparejada el uso de IPC (comunicación entre procesos). Sin embargo, cuando se comparte entre threads, las más simples estructuras de datos se vuelven susceptibles a condiciones de carrera si requieren más de una instrucción de CPU a actualizar: dos threads pueden terminar intentando actualizar una estructura de datos al mismo tiempo. Otra complejidad es que estos bugs ocasionados por estas carreras entre threads son difíciles de aislar y reproducir.
Para prevenir esto, es necesario usar primitivas de sincronización tales como mutex o semáforos, para lockear estructuras de datos contra procesos concurrentes, logrando que solo uno la modifique al a vez. Un thread que se encuentra con un mutex tomado, debe quedarse esperando a que se libere el mismo.



5.3 Mecanismos de sincronización: Events, Mutex y Semaphores.

Event object

Un objeto de acontecimiento es un objeto de sincronización cuyo estado explícitamente puede ser puesto a señalado por el empleo de la función de SetEvent.
El objeto de acontecimiento es útil en el enviar a una señal a un hilo que indica que un acontecimiento particular ha ocurrido. Por ejemplo, en la entrada superpuesta y la salida, el sistema pone un objeto de acontecimiento especificado al estado señalado cuando la operación superpuesta ha sido completada. Un hilo solo puede especificar objetos de acontecimiento diferentes en varias operaciones simultáneas superpuestas, luego usar uno del objeto múltiple esperan funciones para esperar el estado de cualesquiera de los objetos de acontecimiento para ser señalados.

MUTEX
Un objeto de mutex es un objeto de sincronización cuyo estado es puesto a señalado cuando no es poseído por ningún hilo, y no señalado cuando es poseído. Sólo un hilo a la vez puede poseer un objeto de mutex, cuyo nombre viene del hecho que es útil en la coordinación del acceso mutuamente exclusivo a un recurso compartido. Por ejemplo, para impedir a dos hilos para escribir a la memoria compartida al mismo tiempo, cada hilo espera la propiedad de un objeto de mutex antes de la ejecución del código que tiene acceso a la memoria. Después de la escritura a la memoria compartida, el hilo libera el objeto de mutex.

Un hilo usa el CreateMutex o la función de CreateMutexEx para crear un objeto de mutex. El hilo de creación puede solicitar la propiedad que inmediata del mutex se oponga y también puede especificar un nombre para el objeto de mutex. Esto también puede crear mutex sin nombre. Para la información adicional sobre nombres para mutex,

acontecimiento, semáforo, y objetos de temporizador, ven la Sincronización de Interproceso.

Un semáforo posee un contador que lleva la cuenta de los *threads* que se hayan sobre la región crítica. Cada vez que un *thread* pasa por el semáforo este contador se decrementa en una unidad, y la operación contraria ocurre cuando el *thread* libera la región crítica. Al final, cuando todos los *threads* han liberado el semáforo, el contador es el valor máximo especificado en uno de los constructores sobrecargados de la clase Semaphore.

C++

```
#include <windows.h>
#include <stdio.h>

#define MAX_SEM_COUNT 10
#define THREADCOUNT 12

HANDLE ghSemaphore;

DWORD WINAPI ThreadProc( LPVOID );

int main( void )
{
    HANDLE aThread[THREADCOUNT];
    DWORD ThreadID;
    int i;

    // Create a semaphore with initial and max counts of MAX_SEM_COUNT

    ghSemaphore = CreateSemaphore(
        NULL,           // default security attributes
        MAX_SEM_COUNT,  // initial count
        MAX_SEM_COUNT,  // maximum count
        NULL);          // unnamed semaphore

    if (ghSemaphore == NULL)
    {
        printf("CreateSemaphore error: %d\n", GetLastError());
        return 1;
    }

    // Create worker threads

    for( i=0; i < THREADCOUNT; i++ )
    {
        aThread[i] = CreateThread(
                     NULL,       // default security attributes
                     0,          // default stack size
                     (LPTHREAD_START_ROUTINE) ThreadProc,
```

```
                    NULL,       // no thread function arguments
                    0,          // default creation flags
                    &ThreadID); // receive thread identifier

        if( aThread[i] == NULL )
        {
            printf("CreateThread error: %d\n", GetLastError());
            return 1;
        }
    }

    // Wait for all threads to terminate

    WaitForMultipleObjects(THREADCOUNT, aThread, TRUE, INFINITE);

    // Close thread and semaphore handles

    for( i=0; i < THREADCOUNT; i++ )
        CloseHandle(aThread[i]);

    CloseHandle(ghSemaphore);

    return 0;
}

DWORD WINAPI ThreadProc( LPVOID lpParam )
{

    // lpParam not used in this example
    UNREFERENCED_PARAMETER(lpParam);

    DWORD dwWaitResult;
    BOOL bContinue=TRUE;

    while(bContinue)
    {
        // Try to enter the semaphore gate.

        dwWaitResult = WaitForSingleObject(
            ghSemaphore,   // handle to semaphore
            0L);           // zero-second time-out interval

        switch (dwWaitResult)
        {
            // The semaphore object was signaled.
            case WAIT_OBJECT_0:
                // TODO: Perform task
                printf("Thread %d: wait succeeded\n",
GetCurrentThreadId());
                bContinue=FALSE;

                // Simulate thread spending time on task
                Sleep(5);

                // Release the semaphore when task is finished

                if (!ReleaseSemaphore(
```

```
                        ghSemaphore,  // handle to semaphore
                        1,            // increase count by one
                        NULL) )       // not interested in previous count
                {
                    printf("ReleaseSemaphore error: %d\n",
GetLastError());
                }
                break;

            // The semaphore was nonsignaled, so a time-out occurred.
            case WAIT_TIMEOUT:
                printf("Thread %d: wait timed out\n",
GetCurrentThreadId());
                break;
        }
    }
    return TRUE;
}
```

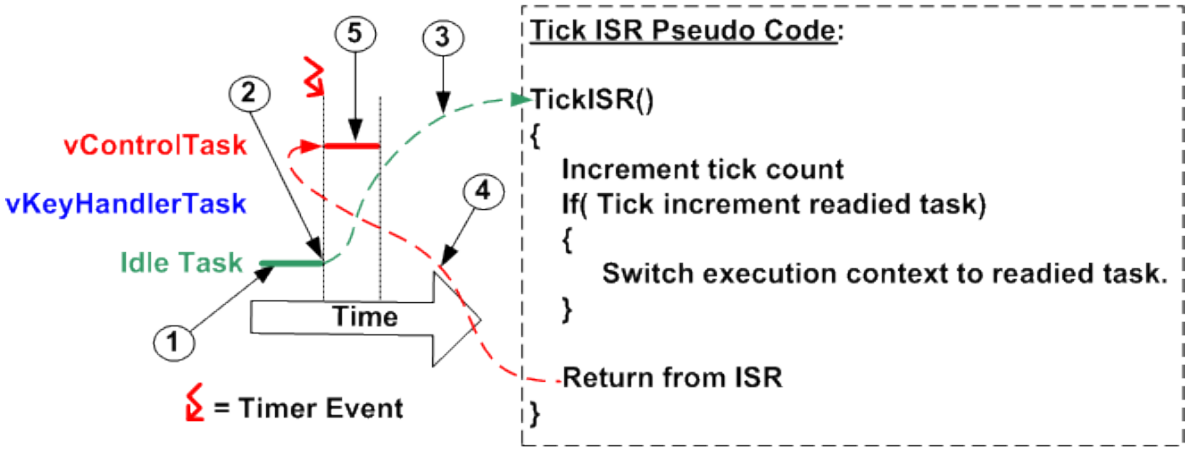
5.4 Mecanismos de sincronizacion: Timers y Critical Sections.

Un objeto de temporizador es un objeto de sincronización cuyo estado es puesto a señalado cuando el tiempo especificado previsto llega. Hay dos tipos de los temporizadores waitable que pueden ser creados: puesta a cero manual y sincronización. Un temporizador del uno o el otro tipo también puede ser un temporizador periódico.

Cuando un temporizador es señalado, el procesador debe correr para tratar las instrucciones asociadas. Temporizadores periódicos de alta frecuencia guardan el procesador continuamente ocupado, que impide al sistema permanecer en un estado de poder inferior para cualquier cantidad de tiempo significativa. Esto puede tener un impacto negativo sobre la duración de la pila de ordenador portátil y los argumentos que dependen de la dirección de poder eficaz, como datacenters grande. Para la eficiencia energética mayor, piense usar notificaciones a base de acontecimiento en vez de notificaciones a base de tiempo en su uso. Si un temporizador es necesario, el empleo un temporizador que es señalado una vez más bien que un temporizador periódico, o poner el intervalo a un valor mayor que un segundo. Un hilo usa el CreateWaitableTimer o la función de CreateWaitableTimerEx para crear un objeto de temporizador. El hilo de creación especifica si el temporizador es tras un temporizador de puesta a cero manual o un temporizador de sincronización. El hilo de creación puede especificar un nombre para el objeto de temporizador. Los hilos en otros procesos pueden abrir una manija a un temporizador existente por especificando su nombre en una llamada a la función de OpenWaitableTimer. Cualquier hilo con una manija a un objeto de temporizador puede usar una de las funciones esperará para esperar el estado de temporizador para ser puesto a señalado.

- El hilo llama la función de SetWaitableTimer para activar el temporizador. Note el empleo de los parámetros siguientes para SetWaitableTimer:
- Use el parámetro lpDueTime para especificar el tiempo en el cual el temporizador debe ser puesto al estado señalado. Cuando un temporizador de puesta a cero manual es puesto al estado señalado, permanece en este estado hasta que SetWaitableTimer establezca un nuevo tiempo previsto. Cuando un temporizador de sincronización es puesto al estado señalado, permanece en este estado hasta que un hilo complete una operación esperaré sobre el objeto de temporizador.
- Use el parámetro lPeriod de la función de SetWaitableTimer para especificar el período de temporizador. Si el período no es el cero, el temporizador es un temporizador periódico; es reactivado cada vez el período expira, hasta que el temporizador sea reinicializado o cancelado. Si el período es el cero, el temporizador no es un temporizador periódico; es señalado una vez y luego desactivado.

Un hilo puede usar la función de CancelWaitableTimer para poner el temporizador al estado inactivo. Para reinicializar el temporizador, llame SetWaitableTimer. Cuando usted es terminado con el objeto de temporizador, llame CloseHandle para cerrar la manija al objeto de temporizador.



Bibliografía

TEORÍA DE MODELOS Y SIMULACIÓN
ARCHIVO PDF
http://www.econ.unicen.edu.ar/attachments/1051_TecnicasIISimulacion.pdf
La definición del sistema y la formulación del modelo
http://books.google.com.mx/books?id=B6LAqCoPSeoC&pg=PA351&lpg=PA351&dq=simulacion%2Bformulacion+del+modelo&source=bl&ots=vO20zdlLH0&sig=9iS9V1x2HqohFLZAcKgFZP1qjUM&hl=es-419&sa=X&ei=tJQkVIn2OILeoASTooGQAg&ved=0CEUQ6AEwBw#v=onepage&q=simulacion%2Bformulacion%20del%20modelo&f=false
http://materias.fi.uba.ar/7526/docs/teoria.pdf
http://es.scribd.com/doc/77587394/Formulacion-de-modelos-de-simulacion
http://www.econ.unicen.edu.ar/attachments/1051_TecnicasIISimulacion.pdf
http://www.anylogic.com/
http://euioscarorgani.blogspot.mx/2013/03/conceptos-basicos-sobre-el-programa.html
https://castalia.forge.nicta.com.au/index.php/en/
http://web.univ-pau.fr/~cpham/WSN-MODEL/Castalia.html
http://www.plm.automation.siemens.com/es_mx/products/tecnomatix/plant_design/plant_simulation.shtml
http://www.libreriaherrero.es/es/libro/9783642050732/bangsow-steffen/manufacturing-simulation-with-plant-simulation-and-simtalk
http://vision-traffic.ptvgroup.com/es/productos/ptv-vissim/casos-de-aplicacion/
http://es.wikipedia.org/wiki/VISSIM

http://ortizol.blogspot.mx/2014/07/receta-csharp-no-4-10-sincronizacion-de-multiples-threads-usando-un-semaforo.html